il NANOBOOK® Z-80™

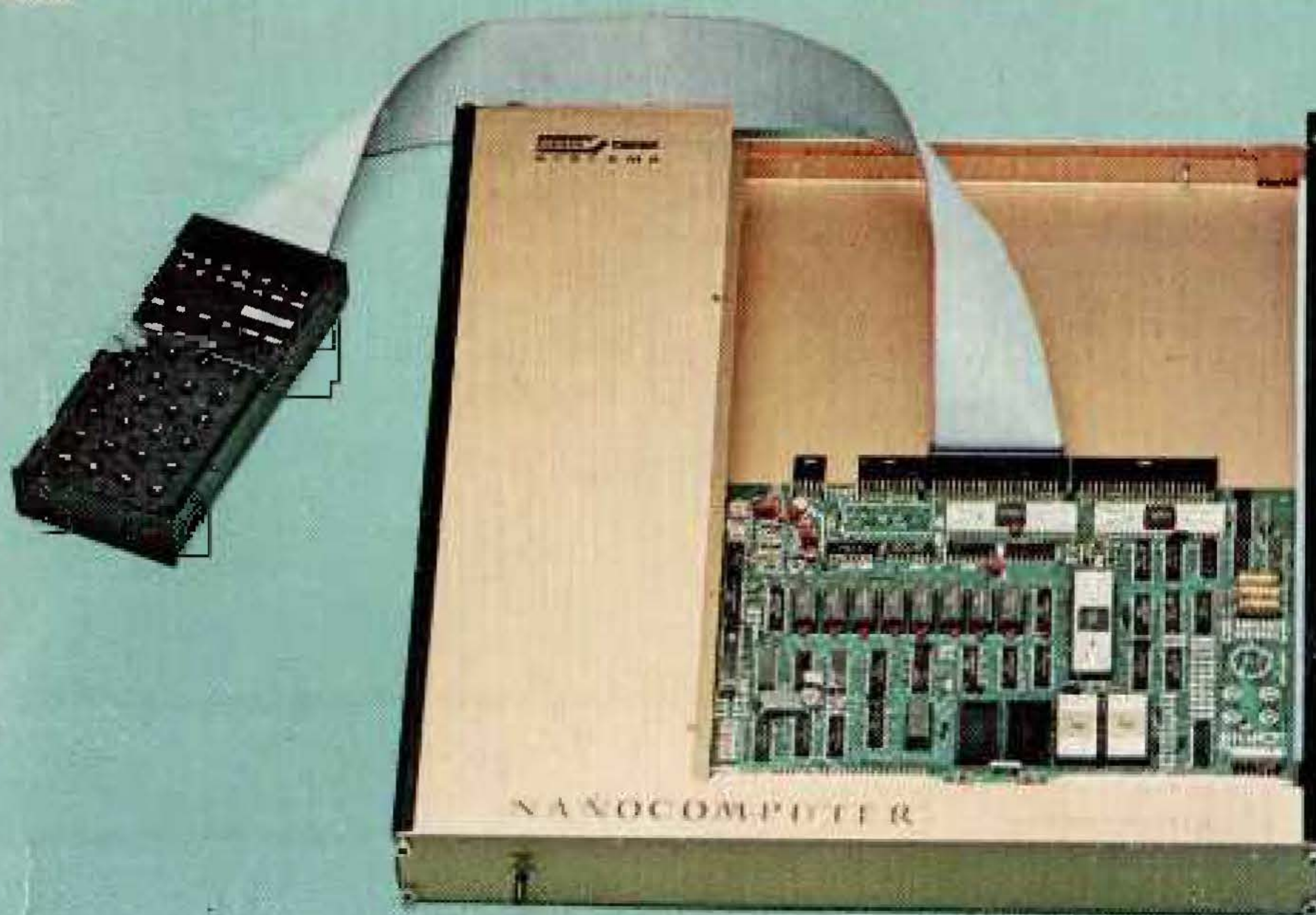
VOL. 1 - TECNICHE DI PROGRAMMAZIONE

EDIZIONE ITALIANA

E. A. NICHOLS
J. C. NICHOLS
P. R. RONY

JACKSON ITALIANA EDITRICE





il NANOBOOK® Z-80™

VOL. 1 - TECNICHE DI PROGRAMMAZIONE

di
Elizabeth A. Nichols, Joseph C. Nichols,
e Peter R. Rony

Versione italiana

Mipro s.r.l.



JACKSON
ITALIANA
EDITRICE s.r.l.

Piazzale Massari, 22 - 20125 Milano

Prima edizione: 1979

Stampato in Italia da
Litografia del Sole - Via Isonzo, 14 - 20094 Buccinasco

PREFAZIONE

I dispositivi elettronici integrati su grande scala hanno determinato una rivoluzione nella concezione degli apparati elettronici, rendendo disponibile a basso costo la flessibilità e la potenza operativa dei calcolatori elettronici.

I calcolatori elettronici, presenti negli anni 50 solo in specializzati centri di calcolo o in centri organizzativi e amministrativi di grandi Enti, sono passati negli anni 60, spesso col nome di minicalcolatori, nei laboratori scientifici e nelle unità produttive delle Industrie, e sono oggi presenti nella forma di microcalcolatori sul tavolo di ogni tecnico, nelle tasche degli studenti, all'interno di prodotti industriali di uso comune, automobili, bilance, registratori di cassa, giochi, ecc..

Questa diffusione, che oggi è esplosiva, determina la necessità di una corrispondente esplosiva diffusione nei tecnici della conoscenza dei meccanismi di funzionamento dei calcolatori elettronici, dei loro linguaggi, delle tecniche di interfacciamento tra essi e gli organi da cui devono prendere informazioni o gli organi che essi devono comandare o che sono destinati a presentare l'informazione da essi elaborata.

La conoscenza degli aspetti strumentali (hardware) e logicali (software) del mondo dei calcolatori è divenuta strumento di lavoro non solo di specialisti ma di ogni ingegnere elettronico e di ogni tecnico a livello intermedio.

I grandi e medi calcolatori potevano essere conosciuti dagli utilizzatori in modo molto mediato, perchè si presentavano "vestiti" con linguaggi evoluti, "Fortran, Cobol, Basic", assai vicini ai linguaggi naturali degli operatori umani; i microcomputer, pur potendosi presentare, nelle forme più evolute, "vestiti", sono molto interessanti quando sono "nudi" e devono essere conosciuti nei loro linguaggi, di macchina. Infatti nella massima parte dei casi i microcomputer, in sè versatilissimi, sono destinati non a divenire dei calcolatori "universali" ma dei calcolatori "dedicati", destinati cioè a un lavoro particolare abbastanza semplice ma da effettuare con la massima efficienza e rapidità: le compilazioni o interpretazioni di linguaggi evoluti, con i relativi tempi di attesa e la minore efficienza dei programmi oggetto così ottenuti, non sono in genere sopportabili.

Bisogna "parlare" al microcomputer col suo linguaggio. Questa necessità è anche un'opportunità per i tecnici, a dir il vero con fatica, di conoscere bene e da vicino i calcolatori.

Ho detto con fatica e non con difficoltà soprattutto presentando questo libro che tratta degli "scarafaggi neri", questi meravigliosi oggetti della tecnica della grande integrazione, dal corpo nero e dai molti piedini, che bisogna imparare ad usare ma che solo pochi specialisti devono saper costruire.

Gli "scarafaggi neri" si imparano ad usare dall'esterno per le loro proprietà funzionali, così come i cavalieri imparano ad usare il cavallo senza pretendere di saperlo costruire, ma quanta arte per cavalcare facendo magari le evoluzioni di squadra!

Questo libro insegna ad usare i microcalcolatori nelle loro unità funzionali: unità centrale, memoria di accesso casuale, memoria di sola lettura, interfaccia di ingresso uscita, senza richiedere al lettore quasi nessuna conoscenza tecnica specialistica, ma presupponendo in lui, semplicemente, capacità logiche.

E' un libro pragmatistico e sperimentale di originale concezione didattica: chi lo legge, eseguendo diligentemente sull'associato Nanocomputer Z-80 gli esercizi, impara a

conoscere il mondo dei microcomputer e i loro linguaggi di macchina e mnemonici (in particolare quelli dello Z-80) e il potente corredo di istruzioni che fa di questi strumenti degli oggetti meravigliosi.

Il novellino è un pò impressionato dal fatto che una macchina così potente come è il suo Nanocomputer richieda tanta programmazione e studio per fare una moltiplicazione in virgola mobile, mentre il suo calcolatorino tascabile Texas o Hewlett Packard fa operazioni a sequenza di operazioni molto più complicate premendo solo pochi tasti, ma va subito avvertito che il Nanocomputer è una macchina "nuda" e se, una volta conosciutala egli la vuole vestire, potrà con molto lavoro di logicale (software) emulare e superare le capacità del suo calcolatore tascabile.

Benvenuto dunque questo libro, e quelli che seguiranno, che apriranno la possibilità a chiunque sia dotato di buona volontà di accedere e partecipare attivamente alla costruzione delle macchine "intelligenti" (ma rispetto all'uomo sempre stupide) del futuro.

Prof. Emilio Gatti
Direttore dell'Istituto di Fisica
del Politecnico di Milano.

Milano, 1979

INTRODUZIONE DEGLI AUTORI

Stiamo assistendo ad una vera e propria rivoluzione nel campo della microelettronica, sempre più impetuosa.

Tutto cominciò trent'anni fa, con lo sviluppo del transistore.

Il transistore, piccolo amplificatore a basso consumo, rimpiazzò la vecchia valvola ad alto assorbimento di potenza, utilizzata dai computer della prima generazione.

Grazie al sinergismo tra i transistori e la logica digitale, le dimensioni ridotte ed il basso costo, questi dispositivi sono divenuti gli elementi costruttivi di base della circuiteria di un computer. I transistori si combinano per formare porte logiche; con queste si possono costruire flip-flop, contatori, sommatori e altre funzioni logiche; questi ultimi, a loro volta, possono essere utilizzati per realizzare memorie, funzioni di controllo e unità aritmetico-logiche (ALU) che, finalmente, sono necessarie per costruire le unità centrali di processo (CPU) di un computer.

Così, il numero di transistori in un circuito logico è misurabile in rapporto alla complessità delle funzioni svolte. Nel 1959 vennero sviluppati su sottili fette di silicio o germanio, i primi circuiti integrati, che consistevano in piccoli gruppi di transistori planari. Con questi, l'era dell'Integrazione a Piccola Scala (SSI) era incominciata: era possibile integrare fino a 12 porte su un solo circuito integrato. Dal '59 in poi, il numero di transistori contenuti nei circuiti integrati più avanzati, è praticamente raddoppiato ogni anno.

Oggi sono disponibili circuiti con 262.144 elementi e l'evoluzione tecnologica è ancora lontana dai limiti teorici.

La CPU Z-80 ed i chip di supporto, introdotti sul mercato dalla Zilog nel 1976, rappresentano lo stato dell'arte nel campo dei microprocessori a 8 bit.

La Zilog sta ora sviluppando il successore dello Z-80: la CPU e i chip di supporto della serie Z-8000. Quest'ultima, tuttavia, sarà una CPU con un'intelligenza paragonabile ad un minicomputer di media capacità: un salto tecnologico significativo. La SGS-ATES, già nota come sorgente alternativa della serie Z-80, sta anch'essa per iniziare lo sviluppo, con tecnologia propria, della serie Z-8000.

Nonostante tutti i progressi fin qui delineati, siamo soltanto all'inizio. Ci si potrà rendere conto realmente di questa rivoluzione, osservando la proliferazione con andamento esponenziale di prodotti e servizi basati sulla microelettronica.

Questo libro è il primo di una serie di volumi sulle tecniche di programmazione e di interfacciamento del microprocessore Z-80.

Il primo volume è dedicato al software dello Z-80: in particolare alla programmazione in linguaggio macchina ed in linguaggio assembler.

Il secondo volume è invece dedicato all'elettronica digitale, mentre il terzo tratta i problemi di interfacciamento con gli elementi CPU, PIO e CTC della famiglia Z-80.

Questi libri sono strutturati come testi-laboratorio e sono pensati in modo da fornire un approccio d'insieme sulla programmazione e sull'interfacciamento di un microcomputer.

Ciò che viene posto in risalto è, in primo luogo, il metodo di apprendere attraverso una continua sperimentazione. Ogni argomento introdotto è approfondito attraverso un lavoro di laboratorio, che permette di verificare non solo l'esattezza o meno degli esercizi svolti, ma anche di evidenziare eventuali errori.

I primi due volumi non richiedono alcuna preparazione specifica su computer, tecniche di programmazione ed elettronica digitale. Il terzo presuppone una familiarità con gli argomenti trattati nei due precedenti.

In tutti e tre i libri la materia viene presentata in un ordine ritenuto dagli autori il più proficuo per l'apprendimento.

Per ogni esercizio è fornita la risposta e ci si è sforzati di prevenire i dubbi derivati dagli esperimenti, individuando ove possibile, l'eventuale estensione logica dei medesimi.

Per rafforzare questo orientamento "di laboratorio" su cui sono costruiti questi libri, abbiamo ritenuto opportuno utilizzare un microcomputer single-board molto sofisticato, basato sulla CPU Z-80, realizzato dalla SGS-ATES: il NANOCOMPUTER®. Esso è un eccellente sistema didattico, poichè, pur semplice ad usarsi per un neofita, è tuttavia dotato di sufficienti requisiti di flessibiità, espandibilità e sofisticazione per tenere vivo anche l'interesse dell'utilizzatore più esperto.

Gli autori sono fortemente riconoscenti a molte persone della SGS-ATES di Agrate Brianza - Milano: R. Baldoni, A. Cattania, D. Comboni, B. Facchi, F. Luraschi, P. Madaschi, C.E. Ottaviani, C. Wallace e soprattutto A. Watts, le cui idee e la cui esperienza sul NANOCOMPUTER®, hanno fortemente arricchito questi libri.

Vorremmo inoltre ringraziare C. Edson e U. Broggi della SGS-ATES in USA, che molto hanno contribuito a facilitare questo progetto, agendo da tramite tra gli USA e l'Italia.

Molto credito è infine dovuto a J. Titus e D. Larsen del Blacksburg Group ed al Dott. Fontana della Microlem Divisione Didattica, per i loro sforzi di coordinamento rispettivamente con la Howard W. SAMS and Co. Inc., per l'edizione americana e con la Jackson Italiana Editrice, per l'edizione italiana; alla societa MIPRO, per la traduzione italiana di questo libro.

Elizabeth A. Nichols
Joseph C. Nichols
Peter R. Rony

® Nanocomputer è un marchio registrato della SGS-ATES

INTRODUZIONE ALL'EDIZIONE ITALIANA

Questo testo sul microprocessore Z-80[*], ideato dagli stessi autori dei primi Bugbook sul microprocessore 8080, si collega a quel particolare e valido filone di ausilio sia didattico che di consultazione tecnica, che si distingue da tutte le altre iniziative del settore per omogeneità e completezza delle trattazioni.

Il lettore che ha già avuto modo di conoscere i Bugbook sul microprocessore 8080 non potrà non apprezzare questo testo, continuazione logica dei precedenti, come il microprocessore Z-80 è sviluppo sul piano tecnologico e della potenzialità operativa del microprocessore 8080.

I concetti tecnici esposti sono specializzati a livello di esperienze pratiche, tramite l'utilizzo esemplificativo dei Nanocomputer NBZ80 e NBZ80-S della SGS-ATES, strumenti didattici di sperimentata efficienza, utilizzati anche dalla società Mipro nei suoi corsi di formazione, sia di base che avanzati.

Nella stesura della versione italiana, si è tenuto conto dell'impostazione e della terminologia già adottata per i precedenti Bugbook.

La rigorosa impostazione tecnico-scientifica, fa di questi testi degli strumenti di base fondamentali per quanti vogliono accostarsi ai microprocessori, in particolare al notissimo microprocessore Z-80[*].

Ringraziamo per la collaborazione prestata il Dr. Baldoni, l'Ing. Reitano, l'Ing. Cattania, il Sig. Ottaviani, il Dr. Comboni e l'Ing. Madaschi della SGS-ATES, la Sig.ra Di Fiore, il Sig. Zanga e il Sig. Reina, della Jackson italiana editrice, la società MIPRO e la Sig. na Daniela Fornari.

Ing. Aldo Cavalcoli — *Milano Giugno 1979*

---

[*] Z-80 è un marchio registrato della Zilog Inc.

SOMMARIO

CAPITOLO 1

CODICI DIGITALI

INTRODUZIONE

Prima di iniziare a programmare il vostro microcomputer, è necessario che conosciate bene come convertire i numeri binari a 8 bit in codice esadecimale, e viceversa, unitamente ad alcuni concetti base sui codici digitali.

OBIETTIVI

Alla fine di questo capitolo sarete in grado di:

- Spiegare cosa significa il termine comunicazione.
- Dare la definizione di bit.
- Dare la definizione di codice binario.
- Dare la definizione di codice digitale.
- Dare la definizione di codice esadecimale.
- Convertire un numero binario a 8 bit in un numero esadecimale a due cifre.
- Convertire un numero esadecimale a due cifre in un numero binario.
- Distinguere tra loro i sistemi di calcolo binari, esadecimali e decimali.
- Elencare diversi codici digitali.
- Elencare diversi dispositivi bistabili.
- Fare un esempio in cui la quantità bit al secondo sia una misura del flusso di informazioni

LINGUAGGI, COMUNICAZIONI E INFORMAZIONE

Una delle più importanti caratteristiche proprie di ogni organismo biologico è la possibilità di comunicare con gli altri organismi della stessa specie. Così la comunicazione, che dà a molti organismi un certo vantaggio per la sopravvivenza, nel senso dato da Darwin al termine, è riscontrabile in molte creature multicellulari, partendo dagli insetti fino all'uomo stesso. Al livello degli insetti esistono molti tipi di comunicazione, che vanno dalla danza delle api ad alcune forme di comunicazione per mezzo di agenti chimici. L'uomo può comunicare con l'ausilio dei suoi cinque sensi, come dimostrano gli individui handicappati che hanno perso uno, o più d'uno, dei loro sensi ma che hanno una sensibilità elevata in quelli rimanenti.

Se supponiamo che un individuo voglia comunicare con un altro per mezzo del senso dell'udito e mediante l'uso della parola, risulta chiaro che esistono molte sfumature rispetto a come un suono pronunciato venga interpretato dall'individuo che lo ascolta. Nei secoli passati, diversi paesi del globo hanno sviluppato ciascuno un proprio sistema convenzionale per interpretare i suoni e trascriverli. Noi chiamiamo questo sistema convenzionale *linguaggio* o anche *linguaggio straniero*. Esistono migliaia di linguaggi diversi, la cui diffusione nel corso dei secoli può aumentare o diminuire, ma di questi solo un numero limitato è oggi utilizzato. Il latino, una volta la lingua dominante in Europa, è considerata oggi una lingua morta, per quanto sia chiara la profonda influenza esercitata su molte lingue europee.

La comunicazione può essere definita come l'impartire, il tramandare, o lo scambiare idee, conoscenze, informazioni etc. (sia per via orale, scritta o con dei segnali). Questa è una delle attività umane più importanti e tipiche. Come puntualizzato da James Martin nel suo eccellente libro, *"Telecomunications and the Computer"*, la capacità delle maggiori *linee di telecomunicazione*, misurata da una quantità detta *bit al secondo*, è avanzata di pari passo con il progresso delle civiltà negli ultimi cento anni. La capacità di queste linee di comunicazione è passata da 1 bit/secondo del 1840 a 50.000.000 di bit/secondo del 1970, raddoppiando cioè ogni 5,08 anni. Martin ha inoltre focalizzato il fatto che il totale delle conoscenze umane è cambiato molto lentamente prima dell'inizio, relativamente recente, del pensiero scientifico. È stato stimato che dal 1800 il totale delle conoscenze è raddoppiato ogni 50 anni; dal 1950 ogni 10; e dal 1970 si raddoppia ogni 5 anni.

Un linguaggio, che può essere considerato come la totalità delle parole e delle combinazioni di parole usate da una nazione, un popolo, o gruppo, non è altro che una forma di comunicazione. I geroglifici egiziani, i simboli e le equazioni matematiche, i segnali di fumo degli Indiani americani, il linguaggio dei segni impiegato dai sordi e l'alfabeto Morse sono altre forme di comunicazione usate dall'uomo.

CODIFICA BINARIA

La "esplosione delle informazioni" avrebbe intasato l'umanità, almeno nei paesi più avanzati, se non si fosse adottato il *sistema di codifica a due stati* per rappresentare tutti i tipi di informazione, cioè i numeri decimali (da 0 a 9), le ventisei lettere dell'alfabeto inglese (dalla A alla Z), operazioni, simboli, regole, e così via. Chiameremo questa codifica col nome di codifica *off-on o codifica binaria*. La codifica binaria può essere rappresentata o espressa da qualsiasi tipo di dispositivo a due stati, come una luce accesa o spenta, un interruttore aperto o chiuso, una scheda meccanografica perforata o non perforata, un nucleo o una parte di nastro o disco magnetici polarizzati "nord" o "sud", due differenti livelli di tensione o corrente, due differenti frequenze; oppure i simboli astratti 0 (off) e 1 (on).

L'importanza del codice binario risiede nel fatto che è possibile costruire dispositivi capaci di cambiare stato molto rapidamente, in tempi minori di 5 ns, o 0,000000005 s. Così un dispositivo può, in linea teorica, trattare, trasmettere o ricevere informazioni alla velocità di 200 milioni di bit al secondo. Trentadue dispositivi di questo tipo che operassero simultaneamente, potrebbero trattare 6,4 miliardi di bit al secondo. È questa possibilità fondamentale che ha permesso alla società di immagazzinare, trattare e comunicare enormi quantità di informazioni.

BIT

L'unità elementare di informazione è chiamata *bit*, abbreviazione di *binary digit*. Potete considerare un bit come una lampadina che può essere accesa (ON) o spenta (OFF). Invece che disegnare lampadine accese o spente, possiamo rappresentare ogni lampadina accesa con il simbolo 1 e ogni lampadina spenta con il simbolo 0.
Così un bit è uguale a una decisione binaria, o ad uno dei due possibili ed egualmente probabili valori o stati (come 0 o 1). Una informazione è generalmente rappresentata da una serie di bit. Per esempio,

1 0 0 0

è la rappresentazione del numero decimale 8 in codice binario. La serie di bit,

1 1 0 0 0 0 0 1

è la rappresentazione della lettera A nel codice a 8 bit ASCII. Tra breve parleremo di questi due codici.

CODICI DIGITALI

Un *codice digitale* è un sistema di simboli che può rappresentare valori di dati e costruire un linguaggio speciale che un elaboratore o un circuito digitale può interpretare e usare. I codici digitali possono essere considerati come "linguaggi" digitali che permettono di immagazzinare, trattare e comunicare informazioni. Come esistono molte lingue parlate così esistono molti codici digitali diversi. I codici possono essere suddivisi in alcune categorie molto importanti:

Categoria I	Codici impiegati dai circuiti elettronici per effettuare operazioni digitali. Esempio: codice binario.
Categoria II	Codici impiegati per convertire i numeri decimali da 0 a 9 in forma digitale. Esempi: codice binario, codice decimale codificato binario (BCD), codice Gray.
Categoria III	Codici impiegati per convertire numeri decimali, le 26 lettere dell'alfabeto inglese, simboli ed operazioni in forma digitale. Esempi: codice ASCII, codice EBCDIC, codice Baudot.
Categoria IV	Codici di istruzioni impiegati da grandi elaboratori, dai mini e dai micro, che permettono loro di eseguire una prescritta sequenza di operazioni. Esempi: codice istruzioni per IBM 370, codice istruzioni per PDP 8/E, codice istruzioni per 8080.

All'interno di questa trattazione esamineremo in particolare quattro codici: codice binario, codice decimale codificato binario (BCD), codice ASCII, e codice di istruzioni per il microprocessor Z-80. Quest'ultimo codice verrà normalmente indicato nel seguito come codice operativo.

CODICE BINARIO

Il più semplice codice digitale è quello a due stati, o codice *binario* che consiste di uno stato 0 (off) e di uno stato 1 (on). Chiameremo questi due stati 0 logico ed 1 logico. Nel codice binario, il decimale 0 è rappresentato dallo 0 logico ed il decimale 1 dall'1 logico. Questa affermazione vi risulterà tra breve più chiara. Come rappresentare allora, i numeri decimali più alti, come 3,17, 586 etc. col codice binario? La soluzione sta nell'usare una serie di bit per formare un sistema di calcolo binario che si fonda sulla *base* o *radice* due. Per esempio, il numero binario 11101_2, dove l'indice 2 rappresenta il sistema di conteggio binario, è equivalente a:

$$11101_2 = (1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 29_{10}$$

dove dovete ricordare che

$2^4 = 16$ in notazione decimale $= 16_{10}$

$2^3 = 8$ in notazione decimale $= 8_{10}$

$2^2 = 4$ in notazione decimale $= 4_{10}$

$2^1 = 2$ in notazione decimale $= 2_{10}$

$2^0 = 1$ in notazione decimale $= 1_{10}$

Allora, $11101_2 = 16_{10} + 8_{10} + 4_{10} + 0 + 1_{10} = 29_{10}$

dove l'indice 10 associato a questi numeri rappresenta il sistema di numerazione decimale, un sistema basato sulla *base* o *radice* 10. Qui di seguito trovate una breve tavola che vi permetterà di convertire numeri decimali semplici in numeri binari:

Numero decimale	Numero binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000

Quindi una serie di digit (vedi nota a piè di pagina) binari, o bit, può rappresentare uno qualsiasi dei sedici numeri decimali da 0 a 15. I numeri decimali maggiori di 15 richiedono

NdT: Nel corso del testo verrà frequentemente utilizzata, in sostituzione della dizione italiana "cifra", la dizione inglese "digit" di uso ormai corrente. Le due dizioni sono comunque del tutto equivalenti.

dei bit addizionali, come mostra la tabella seguente:

Numero decimale	Numero binario
0	0
1	1
2	10
3	11
4	100
7	111
8	1000 = 4 bit
15	1111 = 4 bit
16	10000
31	11111
32	100000
63	111111
64	1000000
127	1111111
128	10000000 = 8 bit
255	11111111 = 8 bit
256	100000000
511	111111111
512	1000000000 = 10 bit
1023	1111111111 = 10 bit
1024	10000000000
2047	11111111111
2048	100000000000
4095	111111111111
4096	1000000000000
8191	1111111111111
8192	10000000000000
16.383	11111111111111
16.384	100000000000000
32.767	111111111111111
32.768	1000000000000000 = 16 bit
65.535	1111111111111111 = 16 bit

Quindi un numero binario di otto *bit*, può codificare 256 numeri decimali diversi, che vanno da 0 a 255_{10}, oppure duecentocinquantasei "entità" diverse, non importa quali esse siano (istruzioni, dispositivi, impulsi, etc.). Dovreste ricordare che lo Z-80 è un microprocessor che ha indirizzi di memoria a 16 bit, parola d'istruzione a 8 bit e parola di selezione dispositivi di I/O a 8 bit. Questo significa che può indirizzare direttamente 65.536 diverse locazioni di memoria, oppure indirizzare 256 diversi dispositivi esterni.

CODICE ESADECIMALE

Ovviamente è difficile ricordare dei numeri espressi in binario. Per esempio, riuscireste a ricordare il seguente numero binario a 8 bit,

1 0 0 1 1 1 0 1

dopo averlo guardato per un solo secondo?
Svelti, copritelo e guardate altrove! Considerate poi il problema di ricordare una lista di numeri ad 8 bit come la seguente:

1 1 0 1 0 0 1 1
0 0 0 0 0 0 1 0
0 0 1 1 1 1 0 0
1 1 0 0 0 0 1 1

Avrete probabilmente concluso che deve esserci un modo migliore per ricordare i numeri binari ad 8 bit. Abbiamo usato i numeri binari ad 8 bit perché li incontrerete di frequente in seguito quando programmerete il microcomputer Z-80. Un sistema per ricordare i numeri binari a più bit è quello di usare il *codice esadecimale*. Il termine *hex* è semplicemente un'abbreviazione della parola esadecimale (in inglese hexadecimal). Il codice esadecimale si riferisce al sistema di numerazione esadecimale, sistema che si fonda sulla base, o radice, 16. Il sistema di numerazione esadecimale consiste di sedici differenti simboli: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Come abbiamo già visto per i numeri decimali, è possibile convertire i numeri esadecimali in numeri binari.

Numero decimale	Numero esadecimale	Numero binario
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111
16	10	0001 0000
17	11	0001 0001
18	12	0001 0010
19	13	0001 0011
20	14	0001 0100
21	15	0001 0101
22	16	0001 0110
23	17	0001 0111
24	18	0001 1000
32	20	0010 0000
40	28	0010 1000
48	30	0011 0000
56	38	0011 1000
63	3F	0011 1111

Abbiamo raggruppato i numeri binari ad 8 bit in due gruppi di 4 bit ciascuno per aiutarvi a comprendere come è stata realizzata la conversione da numero esadecimale a numero binario. Lo spazio tra ciascun gruppo di quattro bit non altera il valore del numero, rende più facile la lettura del numero binario ed è diventata una convenzione standard. Torniamo ora al problema di convertire un numero binario ad 8 bit in codice esadecimale. La procedura per eseguire la conversione è costituita da 3 fasi:

1. Scrivere per intero un numero binario ad 8 bit.

2. Spezzare questo numero binario ad 8 bit in due gruppi di quattro bit ciascuno.

3. Sostituire il corrispondente digit esadecimale

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

per ciascun gruppo di quattro bit.

Eseguendo quanto indicato, realizzate la conversione di un numero binario ad 8 bit in codice esadecimale a due digit. Ciascun gruppo di quattro digit binari è convertito in modo indipendente dall'altro. Ad esempio, si consideri il numero binario ad 8 bit

1 0 0 1 1 1 0 1

Prima, spezzate questo numero in 2 gruppi di quattro bit ciascuno

1 0 0 1 1 1 0 1

Infine, sostituite il digit esadecimale equivalente ad ognuno di questi due gruppi

9 D

Questo è il risultato esatto, $9D_{16}$, dove l'indice "16" rappresenta il sistema di numerazione esadecimale. Di seguito, sono elencati altri numeri esadecimali ed i loro corrispondenti numeri binari ad 8 bit:

Numero decimale	Numero binario		Numero esadecimale
64	0100	0000	40
72	0100	1000	48
73	0100	1001	49
74	0100	1010	4A
96	0110	0000	60
120	0111	1000	78
127	0111	1111	7F
128	1000	0000	80
160	1010	0000	A0
184	1011	1000	B8
191	1011	1111	BF
248	1111	1000	F8
255	1111	1111	FF

NOTA SULLA RAPPRESENTAZIONE DEI NUMERI

Vi può essere capitato che, trattando con tutti questi differenti metodi di rappresentazione dei numeri, binario, esadecimale e decimale, sia sorta la possibilità di confusione. Ad esempio, il numero 10 può essere un numero decimale, esadecimale o binario. Per rimediare a questo problema, ogni volta che c'è qualche possibilità di ambiguità, tutti i numeri esadecimali saranno seguiti dalla lettera H, ad es. 10H, tutti i numeri decimali saranno seguiti da un punto, ad es. 10, e tutti i numeri binari appariranno senza una notazione speciale, ad es. 10 oppure 0110.

DIMOSTRAZIONI

Nei primi due capitoli abbiamo incluso un insieme di esercizi, che abbiamo chiamato dimostrazioni. Queste dimostrazioni sono state realizzate per incoraggiarvi ad utilizzare immediatamente il Nanocomputer, anche se non avete completamente compreso al momento, il Nanocomputer stesso. E' importante che voi eseguiate queste dimostrazioni, anche se avete la sensazione che non stiate facendo altro che premere dei pulsanti, senza sapere il significato dell'operazione.

DIMOSTRAZIONE N. 1

Passo 1

Con riferimento al manuale di istruzioni del Nanocomputer, applicate l'alimentazione al Nanocomputer. Premete il tasto di RESET. Alcuni digit sul display (visualizzatore luminoso) a 7 segmenti dovrebbero accendersi. Altrimenti, premete ancora RESET. Con tale operazione preparate il Nanocomputer ad operare correttamente.

Passo 2

Notate che la tastiera del Nanocomputer possiede due tasti con indicata una freccia. Premete uno di questi tasti alcune volte e verificate cosa succede. Noi abbiamo osservato tre cose. Innanzitutto abbiamo osservato che la luce rossa di selezione circola su undici diverse posizioni possibili. Abbiamo anche osservato che la luce di selezione può essere spostata di un solo passo alla volta premendo e rilasciando rapidamente il tasto; in alternativa, la luce di selezione può essere fatta circolare in modo automatico tenendo premuto il tasto e rilasciandolo quando si accende la luce corrispondente alla locazione desiderata. Come ultima cosa, abbiamo osservato che i digit compaiono sul display concordemente alla posizione della luce di selezione.

Passo 3

Premete l'altro tasto indicato con una freccia ed osservate cosa succede. Noi abbiamo osservato che la luce di selezione circola tra le undici differenti posizioni possibili nella direzione opposta a quella osservata nel Passo 2.

Passo 4

Posizionate la luce di selezione alla posizione indicata con MEM. Prendete nota di quello che appare sui 4 digit più a sinistra del display. Noi abbiamo osservato 0000.

Passo 5

Premete il tasto INC più volte e prendete nota di quanto succede. Noi abbiamo osservato la seguente sequenza di digit in corrispondenza dei 4 digit di sinistra del display:

0000	0001	0002	0003	0004	0005	0006	0007	0008	0009
000A	000b	000C	000d	000E	000F	0010	0011	0012	0013

Notate che viene visualizzata la sequenza di digit esadecimali 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, C, d, E, F, correttamente aggiustati a destra, in ciascun gruppo di 4 digit.

Notate anche che i digit esadecimali A, C, E ed F appaiono come lettere maiuscole, ma che i digit esadecimali b e d appaiono minuscoli. Questo fatto è dovuto a come il display a sette segmenti è utilizzato per rappresentare le lettere, e pertanto nel testo saranno rappresentati con B e D rispettivamente.
I quattro digit esadecimali da 0000 a 000F rappresentano i digit decimali da 0 a 15, il digit esadecimale 0010 rappresenta il numero decimale 16, 0011 rappresenta 17, e così via. Quindi noi abbiamo una tastiera esadecimale. In conclusione, il Nanocomputer è in grado di dialogare con voi tramite la rappresentazione esadecimale dei numeri, quindi voi dialogherete con il Nanocomputer utilizzando la stessa rappresentazione esadecimale.

Passo 6

Premete il tasto RESET. Notate che l'indicazione luminosa si è spostata. Spostate indietro l'indicazione luminosa alla locazione con MEM. Ora abbiamo 0000 ancora una volta visualizzato, e successive pressioni del tasto INC faranno sì che il Nanocomputer visualizzi successivi digit esadecimali. Qual è il numero esadecimale più grande che il Nanocomputer è in grado di visualizzare utilizzando solo questi quattro digit esadecimali? Qual è l'equivalente decimale di questo numero?

Risposta: il numero esadecimale più grande che il Nanocomputer può visualizzare è FFFF. L'equivalente esadecimale è 65.535.

DOMANDE RIEPILOGATIVE

Le seguenti domande servono ad aiutarvi a ripassare i codici digitali.

1. Che cos'è un codice digitale?

2. Elencate vari tipi di codici digitali.

3. Quanti bit ci sono nei seguenti numeri binari?

   a. 11010011
   b. 1000000000000011
   c. 1001

4. A quali numeri decimali corrispondono i seguenti numeri binari?

   a. 11101
   b. 11111111
   c. 1111111111111111
   d. 1001
   e. 11010011
   f. 10011

5. A quali numeri esadecimali corrispondono i seguenti numeri binari?

   a. 11010011
   b. 00111110
   c. 01110110
   d. 00111100
   e. 11111111
   f. 00110010
   g. 11000011
   h. 00000010
   i. 110

6. A quali numeri binari corrispondono i seguenti numeri esadecimali?

   a. D3H
   b. FFH
   c. 32H
   d. 3EH
   e. 76H
   f. 02H
   g. 5H
   h. 3CH
   i. 00H

7. Che cosa significano i seguenti indici?

   a. (16)
   b. (10)
   c. (2)

8. Date la definizione dei seguenti termini:

   a. Sistema di numerazione esadecimale
   b. Bit
   c. Codice binario
   d. Comunicazione
   e. Linguaggio

9. I seguenti numeri sono binari, esadecimali o decimali?

   a. 1111
   b. 1101.
   c. 1100 H

RISPOSTE

1. Un codice digitale è un insieme di simboli che rappresentano valori di dati e formano uno speciale linguaggio che un computer o un circuito digitale può interpretare e usare.

2. Codice binario. Binario decimale codificato. Codice Gray. Codice ASCII. Codice EBCDIC. Codice Baudot. Codice istruzioni IBM 370. Codice istruzioni Z-80.

3. a. Otto
   b. Sedici
   c. Quattro

4. a. 29
   b. 255
   c. 65.535
   d. 9
   e. 211
   f. 19

5. a. D3 H
   b. 3E H
   c. 76 H
   d. 3C H
   e. FF H
   f. 32 H
   g. C3 H
   h. 02 H
   i. 06 H

6. a. 11010011
   b. 11111111
   c. 00110010
   d. 00111110
   e. 01110110
   f. 00000010
   g. 101
   h. 00111100
   i. 00000000

7. a. Si riferisce al sistema di numerazione esadecimale.
   b. Si riferisce al sistema di numerazione decimale
   c. Si riferisce al sistema di numerazione binario

8. a. Un sistema di numerazione basato su una base, o radice, di 16.
   b. Una unità elementare di informazione. E' uguale ad una decisione binaria, o ad uno dei due possibili ed egualmente probabili valori o stati usati per immagazzinare o trasmettere informazioni.
   c. Un codice in cui ciascun elemento può avere solo due stati, in genere espressi come 0 logico e 1 logico.
   d. L'impartire, tramandare o scambiare idee, conoscenze, informazioni, ecc. (sia con la parola, lo scritto o per mezzo di segni).
   e. E' l'insieme della parole e delle combinazioni di parole usate da una nazione, un popolo o un gruppo.

9. a. Binario
   b. Decimale
   c. Esadecimale

CAPITOLO 2

INTRODUZIONE ALLA PROGRAMMAZIONE DEI MICROCOMPUTER

INTRODUZIONE

Nei capitoli seguenti, farete due diversi tipi di esperimenti: (a) esperimenti che richiedono solo la programmazione di microcomputer e (b) esperimenti che richiedono sia la programmazione dei microcomputer che il loro *interfacciamento,* cioè il cablaggio di circuiti che collegano il microcomputer a qualche tipo di dispositivo esterno. Siccome il denominatore comune di tutti gli esperimenti è la programmazione, cominceremo con l'introdurvi ai principi base della programmazione e alle caratteristiche del *linguaggio di programmazione* che userete in questo testo: il set di istruzioni del microprocessor Z-80. Strada facendo definiremo un certo numero di termini importanti, quali *computer, linguaggio macchina, microcomputer* e molti altri. Questa introduzione alla programmazione occuperà un certo numero di capitoli. Infatti abbiamo preferito illustrarvi le nuove istruzioni in gruppi di poche per volta piuttosto che tutte insieme.

OBIETTIVI

Alla fine di questo capitolo sarete in grado di:

- Dare la definizione di *computer digitale.*
- Dare la definizione di *microcomputer.*
- Distinguere tra istruzioni per microcomputer scritte in codice binario, codice esadecimale, o codice mnemonico.
- Distinguere tra rappresentazione mnemonica e linguaggio macchina.
- Dare la definizione di *byte.*
- Convertire un indirizzo di memoria di 16 bit in byte di indirizzo Hi e LO.
- Convertire istruzioni codificate in binario a 8 bit in istruzioni codificate in esadecimale e viceversa.
- Distinguere tra memoria di lettura/scrittura e memoria di sola lettura.
- Dare la definizione di *memoria.*
- Dare la definizione di *programma per un computer.*
- Esprimere le aree delle locazioni di memoria in codice binario o esadecimale, per il vostro microcomputer.
- Identificare i byte a 8 bit in un elenco di numeri binari.

CHE COSA È UN COMPUTER?

Esistono diversi tipi di computer: *computer digitali, computer analogici, computer a fluidi, computer meccanici.* In questi capitoli conoscerete solo *computer digitali,* che costituiscono il 99% di tutti i computer oggi in uso. Un computer digitale può essere definito come segue:

Computer digitale — Dispositivo elettronico capace di accettare, immagazzinare e manipolare aritmeticamente informazioni e che contenga sia i dati che il programma di controllo. L'informazione è trattata nella forma di digit codificati in binario (0 e 1) e rappresentati da due livelli di tensione.

Ogni dispositivo, di solito elettronico, atto ad accettare informazioni, fare confronti, sommare, sottrarre, moltiplicare, dividere e integrare le informazioni, rappresentate in forma di cifre binarie (0 e 1) e fornire i risultati di queste operazioni in una forma comprensibile. Le caratteristiche principali di un computer digitale sono quelle di memorizzare, controllare, eseguire operazioni aritmetiche, logiche e di ingresso/uscita.

Bisogna sottolineare che un computer digitale tratta *informazioni binarie,* del medesimo tipo di quelle di cui abbiamo parlato nel Capitolo 1. L'informazione binaria è di solito nella forma di *codici digitali:* codici istruzioni; codici impiegati per convertire numeri decimali in forma digitale; codici impiegati nella circuiteria eletronica per eseguire le diverse operazioni digitali; e infine codici impiegati per convertire l'alfabeto, i numeri decimali, i simboli e le operazioni in forma digitale.

COSA È UN MICROCOMPUTER?

Un *microcomputer* è un computer digitale completo, basato su un *microprocessor.* Un *microprocessor* è un circuito integrato (chip) che possiede almeno il 75% della capacità di un computer digitale nel calcolo e trattamento dati. Di solito non può funzionare senza l'ausilio di altri chip di supporto e di memoria. Un *circuito integrato* è un dispositivo elettronico nel quale gli elementi attivi (per es. i transistor) e gli elementi passivi (per es. i resistori) sono contenuti all'interno di una singola piastrina di silicio. Nell'elettronica digitale il termine si applica principalmente a circuiti contenenti elementi semiconduttori(2). Il microprocessor è il prodotto della tecologia avanzata dell'industria dei semiconduttori, cioé della capacità che hanno oggi le industrie di costruire migliaia di transistori in un solo chip di silicio non più grosso di 60-80 millimetri quadrati.

COSA È UN PROGRAMMA PER COMPUTER?

Un *programma per computer* può essere definito come una serie di istruzioni o "statements" preparate in una forma comprensibile al computer con lo scopo di conseguire un certo risultato. Questa definizione non pone limitazioni al tipo di risultato che si vuole ottenere. Per esempio, si può essere interessati solo a sistemare i dati in ingresso in una forma più comoda in modo che possano essere sia immagazzinati che usati come uscita. Coi microcomputer sarete però più interessati a scrivere programmi che controllano le operazioni di una macchina o di un dispositivo. In una lavatrice domestica potreste pensare di controllare la quantità di acqua da usare, la temperatura dell'acqua nei diversi cicli di lavaggio, il numero e tipo dei cicli da impiegare per ogni tipo di tessuto e la durata di ogni ciclo. Tutte queste cose possono essere fatte con un programma appositamente e correttamente scritto.

ISTRUZIONI

L'istruzione può essere definita come un insieme (set) di caratteri che definiscono una operazione. Sia singolarmente, che con altre, una istruzione fa sì che un computer digitale esegua una operazione o tratti un determinato dato.

Un *carattere* è un simbolo appartenente a un insieme di simboli elementari come quelli corrispondenti ai tasti di una macchina da scrivere. I simboli comprendono di solito decimali da 0 a 9, le lettere dalla A alla Z, i segni di punteggiatura, il simbolo del dollaro, le virgole, i simboli delle quattro operazioni e ogni altro simbolo che un computer può leggere, memorizzare o scrivere. Nel programmare un computer, non è insolito usare tutta la tastiera della macchina da scrivere, compresi i simboli quali:

@ #, $, %, &, *, (,), /, e eventualmente altri

Le istruzioni si presentano sotto diverse forme. Possono essere rappresentate con:

numeri binari,

11010011
00111110

numeri esadecimali

D3H
3EH

codici mnemonici

OUT 3EH
LDA, 02H

parole complete,

INVIA IL DATO PRESENTE NELL'ACCUMULATORE AL DISPOSITIVO N. 3E (HEX)

CARICA NEL REGISTRO A IL DATO 02 (HEX)

oppure espressioni puramente matematiche

$$X = A^{**}2 + B*Y + C$$

Le istruzioni che userete in questi capitoli saranno rappresentate con numeri binari, esadecimali e in modo mnemonico.

RAPPRESENTAZIONI MNEMONICHE

Mnemonico è un termine che descrive qualcosa usato per aiutare la memoria umana. Da questa definizione ricaviamo le seguenti:

Codice mnemonico — Istruzioni scritte in una forma che il programmatore può facilmente ricordare, ma che può essere facilmente convertita in un linguaggio macchina da un computer o da un utente.

Linguaggio mnemonico — Un linguaggio di programmazione che è basato su simboli facilmente ricordabili e che può essere assemblato in un linguaggio macchina da un computer.

Codice operativo mnemonico, istruzione mnemonica	Istruzioni che sono scritte con notazioni che richiamano la funzione (espressa in inglese) delle istruzioni stesse; per esempio, ADD, MPY, o STO.

In questa trattazione, impiegheremo all'occorrenza i codici mnemonici per le istruzioni che userete nel programmare il microcomputer. I codici mnemonici saranno quelli adottati dalla Zilog Corporation per il set di istruzioni del microprocessor Z-80 prodotto anche dalla SGS-ATES, il quale è composto di 158 diverse istruzioni in linguaggio macchina. Col tempo sarete in grado di effettuare la conversione codice macchina (cioè codice binario) - codice mnemonico e viceversa.

LINGUAGGIO MACCHINA

Un moderno computer digitale elettronico è capace di effettuare manipolazioni usando segnali elettronici binari, di solito due livelli di tensione (+5 volt e massa) che rappresentano rispettivamente gli stati logici 1 e 0. Perciò, ogni istruzione è scritta come una serie di uni e zeri che caratterizza una data istruzione e nessun'altra. Diremo la rappresentazione binaria di una istruzione *linguaggio macchina* o *codice macchina*. Per esempio, l'istruzione in linguaggio macchina 00000111_2 ruota di un bit a sinistra il contenuto di un accumulatore presente nel microprocessor Z-80. L'istruzione 00001111_2, ruota di un bit a destra il contenuto dell'accumulatore.

Dopo questa serie di capitoli sarete in grado di usare le istruzioni in linguaggio macchina del microprocessor Z-80. Queste istruzioni vi saranno proposte in *codice esadecimale*, in modo che possiate ricordarle più facilmente. Alcuni codici esadecimali di istruzioni che userete più frequentemente negli esperimenti di programmazione più semplici sono:

C3H	Istruzione di salto incondizionato
76H	Istruzione di Halt (Alt)
3CH	Istruzione che incrementa di 1 il contenuto dell'accumulatore
3EH	Istruzione di caricamento immediato in accumulatore

In questo paragrafo abbiamo usato alcuni concetti nuovi: *salto incondizionato, caricamento, alt, incrementare* e *accumulatore*. Ne riparleremo tra breve.

UN PROGRAMMA SEMPLICE

Esaminiamo il seguente semplice programma per lo Z-80:

00H	Nessuna operazione (No operation)
3EH	Carica il contenuto della prossima locazione di memoria nell'accumulatore
FFH	Byte di dato
76H	Alt

Il programma contiene tre operazioni e un dato. In questo caso il programma è stato scritto in

codice esadecimale, che avete studiato nel capitolo 1. Ma lo stesso programma poteva esser scritto usando il codice binario, come mostrato qui sotto:

00000000	Nessuna operazione
00111110	Carica il contenuto della prossima locazione di memoria nell'accumulatore
11111111	Byte di dato
01110110	Alt

Alternativamente il programma poteva esser scritto in codice mnemonico e convertito più tardi in codice macchina con l'aiuto di un programma particolare chiamato *assemblatore*. In questo caso avremmo avuto il seguente programma:

NOP	Nessuna operazione
LD A,FFH	Carica il byte FF in accumulatore
HALT	Alt

Notate che il programma in codice mnemonico è composto da parole o da abbreviazioni di parole (in inglese), come NOP, HALT e LD.

Come esegue il suo programma il microcomputer? Lo esegue passo passo e la prima istruzione, NOP, sarà la prima ad essere eseguita. Si avrà quindi una sequenza di operazioni del tipo:

1. Il microcomputer esegue l'istruzione NOP. Non effettua quindi alcuna operazione per un *ciclo di istruzione* e poi passa alla prossima istruzione. L'istruzione NOP ha un uso molto importante, come vedremo in un successivo programma.

2. All'esecuzione dell'istruzione LD, che ha il codice esadecimale 3E, il microcomputer guarda alla successiva locazione di memoria per determinare quale valore deve essere caricato in accumulatore.

3. Il microcomputer va alla prossima locazione di memoria, dove trova FF. Prende questo valore e lo mette nell'accumulatore del microcomputer.

4. Il microcomputer esegue l'istruzione finale HALT. Questa istruzione causa l'arresto del computer.

A seconda dei casi, il programma che abbiamo appena visto può sembrare incomprensibile o semplice. Cos'è la memoria? Cos'è un byte? Come si fa a distinguere un byte di un'istruzione da un byte rappresentante un dato? Cos'è un accumulatore? sono tutte domande ragionevoli ed alcune ve le sarete poste studiando il programma riportato nella pagina precedente.
Adesso proseguiamo col rispondere ad alcune di queste domande.

BYTE

Un *byte* è un gruppo di otto bit contigui che occupano una singola locazione di memoria nel microcomputer Z-80. Per "contigui" intendiamo: adiacenti, vicini, o uno dopo l'altro. Un byte può essere una qualunque delle 256 possibili differenti combinazioni di 8 cifre binari, ciascuno dei quali può essere 0 od 1. Allora, il numero binario

0 1 1 0 1 0 0 1

è un byte, mentre il numero binario

1 0 1 0 0 1

non è un byte, dato che contiene solo sei bit. Il termine byte è ormai molto popolare, in quanto molti computer digitali hanno una lunghezza di parola multipla di 8 bit. Per individuare facilmente i singoli bit nell'ambito di un byte, i bit sono numerati da 0 a 7:

D7	D6	D5	D4	D3	D2	D1	D0

La lettera "D" (che sta ad indicare Dato) è talvolta assente.
Il bit più significativo (MSB) è D7. *Il bit meno significativo* è D0. In genere, per *parola* si intende il numero di bit che un computer può trattare simultaneamente. Se il numero di bit in una parola è otto, si usa di solito il termine byte anziché il termine "parola".
Ad ogni modo, lo Z-80 ha una lunghezza di parola di otto bit. La lunghezza di parola per un minicomputer PDP 8 è dodici bit, il che significa che il minicomputer PDP8 può trattare 12 bit alla volta mentre esegue un programma. Il minicomputer PDP 11 ha una lunghezza di parola di 16 bit, e i grossi computer hanno lunghezza di parola di 32,36 o 60 bit.

MEMORIA

La memoria può essere definita come un dispositivo che è capace di immagazzinare bit, cioè zeri ed uno logici, in un modo che sia poi possibile accedere ad un singolo bit o ad un gruppo di essi[6]. Vi sono molti tipi di memorie che possono realizzare quanto detto, ma nel nostro microcomputer ci sono due tipi di memorie:

Memoria a lettura/scrittura — memoria a semiconduttore, nella quale gli stati logici 0 e 1 possono essere scritti (immagazzinati) e letti (ritrovati). Questa è detta anche memoria ad accesso casuale (random access memory, RAM).

Memoria a sola lettura — memoria a semiconduttore dalla quale i dati, in forma digitale, possono essere ripetutamente letti, ma mai scritti, come nel caso della memoria a lettura/scrittura. La sua abbreviazione è ROM (read-only memory, memoria a sola lettura).

In realtà nel nostro microcomputer, la memoria a sola lettura è un tipo un po' speciale di memoria ed è detta *memoria a sola lettura programmabile e cancellabile* (erasable programmable read-only memory) o EPROM. In un capitolo successivo, parleremo anche delle memorie EPROM.

Il punto veramente importante è che le memorie sono composte di dispositivi a semiconduttore; sono relativamente poco costose, non hanno parti meccaniche e non occupano molto posto sulla scheda del vostro circuito stampato. Sono, in effetti, uno dei ritrovati che hanno permesso il rapido evolversi della tecnologia dei computer.

Quanta memoria avete a disposizione? Il più semplice Nanocomputer che potete usare ha 4096 byte di memoria a lettura/scrittura e 2048 byte di memoria a sola lettura. Siccome un byte contiene otto bit, avete a disposizione nel vostro microcomputer un totale di 49.152 bit di memoria e ciò è sufficiente, come vedremo nei prossimi capitoli, per esperimenti di programmazione e di interfacciamento.

Sul vostro Nanocomputer siete in grado di aumentare il numero dei byte della memoria a lettura/scrittura fino a 16.384, ed il numero dei byte della memoria a sola lettura fino a 8192. Un Nanocomputer con questa capacità di memoria ha un totale di 24.576 byte di memoria. Esso è conosciuto come microcomputer con 24K, dove la "K" rappresenta circa mille (1024 per l'esattezza) diverse locazioni di memoria. Un microcomputer con 4K conterrebbe 4096 byte di memoria. Un microcomputer con 64K conterrebbe 65.536 byte di memoria.

INDIRIZZI DI MEMORIA

Un *indirizzo di memoria* è definito come la locazione in cui viene immagazzinata una parola di memoria. Notate che abbiamo detto parola e non byte; infatti per alcuni computer, la parola può essere di 32 bit, e allora ogni locazione di memoria conterrà 32 bit. Nel microcomputer Z-80 ogni locazione di memoria contiene un byte, cioè otto bit.

Nel Nanocomputer standard che userete ci sono 6144 diverse locazioni possibili, se consideriamo la versione con la capacità standard. Questa memoria è suddivisa in 2 gruppi:

- Gruppo di memoria 1: il primo gruppo di 4096 (4K) locazioni di memoria, ciascuna delle quali contiene 8 bit. Questa è la memoria di lettura/scrittura che utilizzerete normalmente nel programmare il microcomputer.

- Gruppo di memoria 2: il secondo gruppo di 2048 (2K) locazioni di memoria, ciascuno delle quali contiene 8 bit. Questa zona di memoria è occupata da memoria a sola lettura, od anche da memoria EPROM (a sola lettura, ma programmabile e cancellabile), che contiene il sistema operativo che gestisce il Nanocomputer. NON POTETE MODIFICARE IL CONTENUTO DI QUESTA ZONA DI MEMORIA.

ESTENSIONE DELLA MEMORIA

Il microcomputer Z-80 ha una proprietà molto importante: può indirizzare fino a 65.536 diverse locazioni di memoria, ciascuna delle quali contiene otto bit. Questo perchè il chip ha la parola di indirizzo di 16 bit. Se fate bene i calcoli, infatti, si possono ottenere 2^{16}, cioè 65.536 combinazioni diverse.

Come abbiamo già detto, il Nanocomputer ha solo 6144 (6K) locazioni di memoria sulla scheda base. Vi chiederete quindi quali siano, tra le 65.536 indirizzabili, quelle disponibili. La risposta è: le prime 4K locazioni, e le ultime 2K.

Detto in un altro modo, l'insieme degli indirizzi di memoria possibili nel Nanocomputer con memoria standard è:

da 0000000000000000_2 a 0000111111111111_2 di memoria lettura/scrittura
da 1111100000000000_2 a 1111111111111111_2 di ROM.

Questa è però una notazione scomoda e molto difficile da ricordare. Ci sono però sistemi più semplici per identificare le locazioni di memoria e l'insieme di quelle disponibili nel vostro microcomputer, e li vedremo in seguito.

INDIRIZZI DI MEMORIA HI e LO

E' molto difficile ricordare gli indirizzi di memoria a sedici bit. Indubbiamente più difficile dei codici istruzione o dei dati, che sono di solo otto bit. Per ricordare questi tipi di indirizzi bisogna pensare che il *microprocessor Z-80 tratta gli indirizzi di memoria a sedici bit come due byte d'indirizzo di otto bit, un byte di otto bit HI e un byte di otto bit LO.* La loro definizione è la seguente:

Byte d'indirizzo HI Sono gli otto bit più significativi (i primi 8 da sinistra) dei sedici della parola d'indirizzo del microprocessore Z-80. La sua abbreviazione è H o HI (dall'inglese "high", alto).

Byte d'indirizzo LO Sono gli otto bit meno significativi (gli ultimi 8 da sinistra) dei sedici bit della parola d'indirizzo del microprocessore Z-80. La sua abbreviazione è L o LO (dall'inglese "low", basso).

Quindi l'insieme delle locazioni di memoria lettura/scrittura indirizzabili nel Nanocomputer 4K è:

da HI $= 00000000_2$ a HI $= 00001111_2$
da LO $= 00000000_2$ a LO $= 11111111_2$

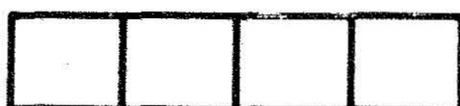
Visto che avete imparato a convertire i numeri binari a otto bit i numeri esadecimali a due digit, fate la conversione degli indirizzi di memoria HI e LO che abbiamo visto otterrete il seguente insieme di locazioni di memoria lettura/scrittura:

da HI $= 00_{16}$ a HI $= 0F_{16}$
da LO $= 00_{16}$ a LO $= FF_{16}$

Ricordate la seguente regola: *per individuare una locazione di memoria, dovete specificare sia il byte d'indirizzo HI che il byte d'indirizzo LO, i quali, insieme, formano una parola di indirizzo di sedici bit.*

DIMOSTRAZIONE N. 2

Ad ogni passo di questa dimostrazione, dovreste avere l'indicazione luminosa di selezione localizzata alla posizione MEM. Notate che vi sono quattro digit esadecimali visualizzati sul lato sinistro del display, ed altri due digit esadecimali visualizzati sul lato destro. I quattro digit di sinistra rappresentano l'indirizzo di una locazione di memoria. I due digit di destra rappresentano il contenuto della locazione di memoria il cui indirizzo è visualizzato sulla sinistra.


INDIRIZZO


DATO

Potete immaginare la memoria come un insieme di scatole. Ciascuna scatola possiede un'etichetta permanente. Queste etichette non sono altro che i numeri esadecimali a partire da 0000, 0001, 0002, 0003 e così via. Dentro ciascuna scatola potete inserire esattamente un byte di informazione sotto forma di due digit esadecimali. Con la tastiera del Nanocomputer potete esaminare le singole locazioni di memoria e modificare il contenuto di una data locazione.

Passo 1

Posizionate l'indicatore luminoso su MEM. Noi abbiamo osservato che l'indirizzo di memoria visualizzato nei 4 digit di sinistra, detto ADDRESS DISPLAY (visualizzatore degli indirizzi), era 0000. Il contenuto della locazione di memoria 000, visualizzato nei due digit più a destra, detto DATA DISPLAY (cioè visualizzatore dei dati), era 00. Modifichiamo il contenuto di 0000 da 00H a 23H. Premete il tasto esadecimale 2, poi il tasto 3 ed infine premete il tasto ST. ST è la notazione abbreviata per la parola STORE (cioé carica). Premendo ST, caricherete (STORE) 23H nella locazione 0000H. Notate che ora l'indirizzo si è automaticamente incrementato di 1, a 0001. Così state ora osservando la locazione di memoria 0001. Caricate 24H (esadecimale) in questa locazione.

Passo 2

Ora esaminate le locazioni 0000H e 0001H per verificare che avete realmente memorizzato 23H (esadecimale) e 24H (esadecimale) in esse. Premete 0, 0, 0, 0, in sequenza sulla tastiera e poi il tasto LA. La è la notazione abbreviata di LOAD ADDRESS (cioé carica l'indirizzo).
Avete caricato così l'indirizzo 0000 nell'address display (visualizzatore degli indirizzi). 0000H. Premete il tasto INC. L'indirizzo di memoria si incrementa di uno e potete verificare che il contenuto della locazione di memoria 0001H è 24H. Dovreste ora essere in grado di osservare il contenuto di qualsiasi locazione e modificare il contenuto di qualsiasi locazione di memoria lettura/scrittura.

Passo 3

Guarderete ora il il contenuto della locazione di memoria ROM (Read Only Memory) e cercherete di scrivere nella Read Only Memory. Guardate il contenuto della locazione FC00H (premete F, C, 0, 0 in sequenza, e poi LA). Noi abbiamo visto che il contenuto di FC00 era 31H. Cercate di porre il valore esadecimale FFH in questa locazione di memoria (premete F, F e poi il tasto ST). Esaminate ora il contenuto di FC00H. Noi abbiamo osservato che il contenuto di FC00H non era cambiato, cioè era ancora 31H. Quindi non siamo stati in grado di scrivere nella memoria a sola lettura ROM.

DOMANDE RIEPILOGATIVE

Le seguenti domande servono ad aiutarvi a ripassare i concetti introduttivi alla programmazione di un microcomputer.

1. Dite, delle seguenti istruzioni, se sono in codice binario, esadecimale o mnemonico.

   a. HALT
   b. 11010011
   c. 3E
   d. LD
   e. INC
   f. 00111100
   g. 76

2. Scrivete le seguenti istruzioni binarie in codice esadecimale:

   a. 11010011
   b. 01110110
   c. 00111100
   d. 00110010
   e. 00000000
   f. 11000011
   g. 11111111

3. Quale delle seguenti espressioni è un byte?

   a. 1001
   b. 011
   c. 0000001100000011
   d. 1110001101
   e. 111000
   f. 0100110

4. Scrivete i seguenti indirizzi di memoria a 16 bit come byte esadecimali HI e LO.

   a. 0000001111111111
   b. 0000000011111111
   c. 0000000111111111
   d. 0000001011111111
   e. 0000000000000000
   f. 0000000100000000
   g. 0000001000000000
   h. 0000001100000000

5. Quali delle seguenti istruzioni sono in linguaggio macchina?

   a. NOP
   b. HALT
   c. LD
   d. INC
   e. 3E
   f. 76
   g. 11010011
   h. 00
   i. 00111100

6. Scrivete l'estensione dei seguenti gruppi di memoria di un microcomputer Z-80, esprimendoli in termini di byte di indirizzo HI e LO.

   a. I primi 4K byte di memoria (lettura/scrittura)
   b. I primi 16K byte di memoria (lettura/scrittura)
   c. Gli ultimi 4K byte di memoria (sola lettura)
   d. Gli ultimi 8K byte di memoria (solo lettura)

7. Date la definizione dei seguenti termini:

   a. Byte
   b. Indirizzo di memoria
   c. Codice mnemonico

RISPOSTE

1. a. Codice mnemonico
   b. Codice binario
   c. Codice esadecimale
   d. Codice mnemonico
   e. Codice mnemonico
   f. Codice binario
   g. Codice esadecimale

2. a. D3
   b. 76
   c. 3C
   d. 32
   e. 00
   f. C3
   g. FF

3. Nessuna delle espressioni elencate contiene otto bit, quindi nessuna di esse è un byte.

4. a. HI = 03 LO = FF
   b. HI = 00 LO = FF
   c. HI = 01 LO = FF
   d. HI = 02 LO = FF
   e. HI = 00 LO = 00
   f. HI = 01 LO = 00
   g. HI = 02 LO = 00
   h. HI = 03 LO = 00

5. Le istruzioni e, f, g, h e i sono in linguaggio macchina.

6. a. L'estensione è da HI = 00 e LO = 00 a HI = 0F e LO = FF
   b. L'estensione è da HI = 00 e LO = 00 a HI = 3F e LO = FF
   c. L'estensione è da HI = F0 e LO = 00 a HI = FF e LO = FF
   d. L'estensione è da HI = E0 e LO = 00 a HI = FF e LO = FF

7. a. Un gruppo di otto bit contigui che occupano una sola locazione di memoria nel microcomputer Z-80.

   b. La locazione in cui viene immagazzinata una parola in memoria.

   c. Istruzione per computer scritta in una forma che il programmatore possa facilmente ricordare e che, in seguito, il computer stesso possa facilmente convertire il linguaggio macchina.

CAPITOLO 3

ALCUNE ISTRUZIONI DEL MICROCOMPUTER Z-80

INTRODUZIONE

In questo capitolo, definiremo alcuni importanti termini quali: *operazione, byte di dati, byte di indirizzo, codice di dispositivo,* e introdurremo alcune semplici istruzioni del microcomputer Z-80, istruzioni che userete nei programmi proposti nel Cap. 5. Il nostro obiettivo è quello di introdurre gradualmente l'intero set di istruzioni dello Z-80 e proporre programmi che vi permettano di verificare l'utilizzo di questi tipi di istruzioni.

OBIETTIVI

Alla fine di questo capitolo sarete in grado di:

* Dare la definizione di programma di un computer
* Dare la definizione di operazione
* Rappresentare, in modo semplice, istruzioni ad un byte, due byte e tre byte
* Spiegare come operano 5 istruzioni del microcomputer Z-80, molto comuni: NOP; HALT; INC A; LD A, data; e JP addr.
* Fare un elenco dei due gruppi di registri generali, e dei sei registri specializzati del microprocessore Z-80
* Fare un elenco dei registri generali che vengono usati a coppie
* Dare la definizione di accumulatore
* Dare la definizione di incremento
* Spiegare come operano 5 istruzioni del microcomputer Z-80, molto comuni: NOP; HALT; INC A; LD A, data; e JP addr.
* Dare le definizione di modo di indirizzamento immediato
* Elencare, per un microcomputer Z-80 a 2,5 MHz, i tempi di esecuzione delle seguenti istruzioni: NOP, HALT, INC A, LDA, n e JP nn.

CHE COSA E' UN PROGRAMMA PER UN COMPUTER?

Un *programma per un computer* può essere definito come una sequenza di istruzioni, le quali, prese nel loro insieme, fanno sì che il computer compia il task (compito) voluto. Cos'è un task? Un task è una operazione o un insieme di operazioni che possono essere eseguite da un computer, tenuto conto della memoria e dei dispositivi di ingresso/uscita disponibili. I programmi vengono immagazzinati in memoria sotto forma di una sequenza di 0 e di 1 (i bit) che il computer può leggere, interpretare, ed eseguire in sequenza, uno alla volta. Per lo Z-80, questi bit sono memorizzati in gruppi di 8 bit detti byte. Una sola istruzione può occupare uno, due, tre o quattro byte consecutivi in memoria. Lo Z-80 esegue un programma leggendo un'istruzione, interpretando le configurazioni dei bit, e poi svolgendo il task necessario per completare l'operazione definita dall'istruzione. Vengono quindi lette le locazioni di memoria successive, finchè non si arriva ad un'istruzione che dice al computer di fermarsi o di saltare ad un'altra locazione di memoria per eseguire l'istruzione successiva.

I programmi non sono costituiti solo da byte di istruzione. Per fornire le informazioni necessarie, nei programmi devono essere inclusi anche dei byte di dati. Per esempio, un programma che debba sommare due numeri, deve contenere i numeri che devono essere sommati fra di loro (byte di dati) nonchè le istruzioni per eseguire l'operazione di addizione (byte di istruzione). Altri tipi di byte presenti in un programma sono i byte di indirizzo, i byte del codice di dispositivo, e i byte di displacement (spiazzamento). Ne parleremo più avanti in questo stesso capitolo.

La configurazione minima di un Nanocomputer fornisce all'utente 4K byte di memoria di lettura/scrittura per la memorizzazione dei programmi. Ciò è sufficiente per memorizzare programmi molto complessi. Vi sono due termini importanti usati per definire i programmi di un computer: essi sono "istruzione" e "operazione". Ora ne spiegheremo il significato.

ISTRUZIONI E OPERAZIONI

Un'*istruzione* è un insieme di caratteri che definiscono, da soli o con altre informazioni, una operazione e che, complessivamente fanno sì che il computer esegua l'operazione indicata. *Un'operazione* è definita come una specifica azione che il computer eseguirà quando un'istruzione gli dirà di farlo (per es. divisione, addizione, sottrazione, esecuzione di OR, ecc.). Il numero di operazioni differenti che un computer può eseguire e la velocità con cui le esegue, sono un indice della "potenza" del computer stesso.
Le operazioni che il microcomputer Z-80 esegue possono essere suddivise nei seguenti gruppi:

- Gruppo di trasferimento dati
- Gruppo aritmetico e logico
- Gruppo di rotazione e di shift
- Gruppo di manipolazione dei bit
- Gruppo di salto, di chiamata e salto a subroutine (Call) e di ritorno da subroutine (Return)
- Gruppo di I/O e di controllo macchina

ISTRUZIONI A PIU' BYTE (MULTI-BYTE)

Molte istruzioni, appartenenti al set dello Z-80, richiedono un solo byte, mentre alcune sono formate da due, tre o anche quattro byte. Chiameremo queste ultime, *istruzioni multi-byte.* Le corrispondenti definizioni sono:

Istruzione a un solo byte	Istruzione che è definita da otto bit contigui che occupano una sola locazione di memoria.

Istruzione a due, tre, quattro byte	Istruzione che è definita da informazioni che occupano due, tre, quattro successive locazioni di memoria.

Il numero di byte richiesti da un'istruzione è strettamente correlato alla complessità dell'istruzione stessa e alle informazioni di cui ha bisogno. Il set di istruzioni dello Z-80 è un ampliamento del set di istruzioni del microprocessor 8080, costruito dalla Intel Corporation. Per mantenere una certa coerenza fra i due set di istruzioni, è stato necessario ricorrere a compromessi nella definizione delle nuove istruzioni dello Z-80. Come risultato, la struttura delle istruzioni dello Z-80 è un pò più complicata di quella dell'8080. Comunque, questo sacrificio è più che compensato dal fatto che tutti i programmi scritti per l'8080 possono essere eseguiti su di un microprocessor Z-80 senza apportarvi alcuna modifica.
Il microprocessore 8080, storicamente, è un microprocessore molto importante, per il quale esiste già moltissimo software.
Quindi, questa "compatibilità verso l'alto" risulta particolarmente interessante. Per una discussione completa sulle analogie e le differenze tra questi due tipi di microprocessori, vi rimandiamo all'appendice.

Vi diamo qui di seguito delle semplici rappresentazioni di istruzioni dello Z-80 a uno, due, tre e quattro byte. Notate che in tutte le istruzioni tranne che in una a quattro byte, il primo o i primi due byte sono codici operativi che specificano quello che l'istruzione fa, mentre gli ultimi byte sono le informazioni necessarie per portare a termine l'esecuzione dell'istruzione. Di questo parleremo più dettagliatamente quando vi ripresenteremo le istruzioni specifiche. I formati delle istruzioni vengono qui presentati semplicemente come anticipazione di un discorso futuro. Dato che i codici operativi si susseguono sequenzialmente in memoria, noi li scriveremo in colonne verticali come una tabella, a differenza delle pagine che state leggendo, che sono scritte in orizzontale.

Le istruzioni ad un solo byte richiedono solo un codice operativo e non necessitano di nessuna informazione ausiliaria:

Codice operativo

Le istruzioni a due byte hanno quattro forme:

Codice operativo	*Codice operativo*	*Codice operativo*
Codice operativo	Byte di dati	Codice di dispositivo

Codice operativo
Byte di displacement (spiazzamento).

Spiegheremo poi brevemente il significato dei termini *"Byte di dati"*, *"Codice di dispositivo"* e *"Byte di displacement"*.

Le istruzioni a tre byte hanno tre forme:

Codice operativo	*Codice operativo*	*Codice operativo*
Byte di dati	Byte d'indirizzo LO	*Codice operativo*
Byte di dati	Byte d'indirizzo HI	Byte di displacement

Abbiamo già chiarito i concetti di byte di indirizzo di memoria LO e HI.

Le istruzioni a quattro byte hanno quattro forme:

Codice operativo	*Codice operativo*	*Codice operativo*
Codice operativo	*Codice operativo*	*Codice operativo*
Byte di dati	Byte d'indirizzo LO	Byte di displacement
Byte di dati	Byte d'indirizzo HI	Byte di dati

Codice operativo
Codice operativo
Byte di displacement
Codice operativo

Come potete facilmente immaginare, gli ultimi due tipi di istruzioni a quattro byte sono istruzioni molto complicate. Più avanti parleremo dettagliatamente di queste istruzioni, facendovi parecchi esempi.

TIPI DI INFORMAZIONI IMMAGAZZINATE IN MEMORIA

La memoria di un microcomputer Z-80 è costituita da una sequenza di locazioni di memoria di otto bit ciascuna. Quando accede alla memoria, il microprocessore opera sempre trattando otto bit per volta. I tipi di informazione che possono essere immagazzinati nella memoria sono:

Codici operativi a otto bit
Byte di dati a otto bit
Codici di dispositivo a otto bit
Byte d'indirizzo LO a otto bit
Byte d'indirizzo HI a otto bit
Byte di displacement a otto bit.

Perciò, in un programma per il microcomputer Z-80, vengono immagazzinati contemporaneamente nella stessa memoria codici di istruzione, byte di dati, codici di dispositivo, byte di indirizzo, e byte di displacement. E' quindi ragionevole domandarsi come il microcomputer sia capace di distinguerli l'uno dall'altro.
La risposta è che l'ordine con cui appaiono le informazioni, definisce il tipo dell'informazione stessa. Programmare un calcolatore è un'attività di precisione: un solo errore e il programma non è più in grado di funzionare correttamente. Un programma in un microcomputer inizia a funzionare ad un dato indirizzo di memoria e procede, una operazione dopo l'altra, fino all'indirizzo di memoria finale. I codici operativi nel programma dicono cosa aspettarsi; cioè, se il prossimo byte è un byte di dati, byte di indirizzo, byte di codice di dispositivo, o un altro codice operativo.

CODICE OPERATIVO

Il primo byte di un'istruzione del microprocessore Z-80 è sempre un *codice operativo*. Notate che alcune istruzioni iniziano con due byte di codice operativo. Queste istruzioni sono ampliamenti del vecchio set di istruzioni dell'8080. Se il primo byte di un'istruzione è CB, DD, ED o FD, anche il secondo byte deve essere un codice operativo. I byte di questo codice definiscono l'azione specifica che il microcomputer Z-80 deve eseguire. Questa azione specifica potrebbe essere un trasferimento dati, un'operazione aritmetica, logica, di salto, di stack, di I/O o di controllo. Se desiderate conoscere cosa farà il microcomputer, dovete guardare il codice operativo dell'istruzione. Un sinonimo di codice operativo è *codice istruzione*.

BYTE DI DATI

Un *byte di dati* è definito come un numero binario a otto bit che il microprocessore Z-80 usa nelle operazioni aritmetiche o logiche, o che immagazzina in memoria. Gli otto bit possono essere in un qualsiasi tipo di codice digitale: codice binario, binario decimale codificato, codice ASCII, ecc. Quando usiamo il termine byte di dati, intendiamo dire che gli otto bit non sono un codice operativo, un indirizzo di memoria, o un codice di identificazione di dispositivo.

Quando programmerete il microcomputer, vedrete che è molto più conveniente mettere dei dati sparsi nel programma, quando e dove volete, piuttosto che far riferimento a locazioni di memoria, magari lontane per gli otto o sedici bit che vi servono.

CODICE DI DISPOSITIVO

Il *codice di dispositivo*, per un microcomputer Z-80, è il codice di uno specifico dispositivo di ingresso o uscita (Input-Output, abbreviato I/O) col quale si vogliono scambiare gli otto bit di informazione e l'impulso di selezione dispositivo. In seguito entreremo più in dettaglio. E' importante notare che si tratta di un codice a otto bit, il che dà la possibilità di indirizzare 2^8, cioè 256 diversi dispositivi di uscita. Nel vostro microcomputer, i codici di uscita da 04 a 07 sono riservati al sistema operativo del Nanocomputer.

Avanzando nella lettura di questo testo, vi consigliamo di studiare molto bene il significato di codice di dispositivo e di impulso di selezione dispositivo, e il modo di usare quest'ultimo per far funzionare i dispositivi di ingresso/uscita, utilizzando il programma del microcomputer.

BYTE DI INDIRIZZO HI E LO

Vi ricordiamo ancora una volta che il *byte di indirizzo HI* è composto dagli otto bit più significativi, i bit di valore più alto, e che il *byte di indirizzo LO* è composto dagli otto bit meno significativi, i bit di valore più basso, dei sedici bit che indirizzano la memoria nel microprocessor Z-80. Siccome lo Z-80 è un microprocessore a otto bit, che prende dati e istruzioni dalla memoria a blocchi di otto bit per volta, non si può trattare l'indirizzo di memoria di sedici bit se non come una coppia di due byte di indirizzo di otto bit.

BYTE DI SPIAZZAMENTO

I byte di spiazzamento (displacement) compaiono nelle istruzioni che usano l'*indirizzamento indicizzato.* L'indirizzamento indicizzato è una tecnica che serve a definire un indirizzo di memoria a due byte aggiungendo uno *spiazzamento* ad un numero a 16 bit che si trova in una speciale locazione del microcomputer chiamata *registro indice.* Uno *spiazzamento* è un *numero con segno, complemento a due.* Non staremo ora a dare la definizione di "complemento a due con segno". E' sufficiente dire che è un metodo di rappresentazione dei numeri binari che facilita la manipolazione dei numeri negativi.

Questo lo spiegheremo molto meglio più avanti. Non scoraggiatevi se molti dei termini che citeremo non vi sono familiari. Riuscirete a capirli completamente solo dopo che avrete acquisito esperienza nell'uso delle istruzioni del vostro Z-80.

CHE COSA E' UN REGISTRO?

Un *registro* è un circuito di memorizzazione temporanea la cui capacità è di solito uguale ad una parola del computer. Nel microprocessore Z-80 ogni singolo registro memorizza un solo byte, cioè 8 bit.

All'interno dello Z-80 esiste un certo numero di registri, alcuni dei quali vengono usati per memorizzare dati e altri vengono invece usati esclusivamente per eseguire le istruzioni. In genere possiamo suddividere i registri in due gruppi: quelli che potete indirizzare da programma e quelli che non potete indirizzare. I registri indirizzabili da programma sono:

— Due gruppi di *registri di uso generale* (non specializzati) a otto bit indirizzabili singolarmente o per coppie,

Gruppo 1:	registro B	*Gruppo 2:*	registro B'
	registro C		registro C'
	registro D		registro D'
	registro E		registro E'
	registro H		registro H'
	registro L		registro L'

Il gruppo 2 è il "set di registri alternativi" (ARS).

— Un *accumulatore* a 8 bit per ogni gruppo, indicati anche come registri A e A'

— Un registro di *flag* per ogni gruppo, detti anche registri F e F'

— Il registro *stack pointer* a 16 bit (SP)

— Il registro *contatore di programma* a 16 bit (PC)

— Due *registri indice* di 16 bit (IX) e (IY)

— Il registro *interrupt page address* (I) a 8 bit

— Il registro *memory refresh* (R) a 8 bit

Questi sono i soli registri con i quali potete scambiare direttamente informazioni mediante un programma appositamente scritto.

REGISTRI NON SPECIALIZZATI

I due gruppi di sei registri generali - B, C, D, E, H e L, e B', C', D', E', H', e L', - memorizzano temporaneamente un solo byte di informazione e, dato che si trovano tutti all'interno del microprocessore Z-80, lo scambio di informazioni tra due di essi è molto rapido, come è rapido anche lo scambio di informazioni tra uno di essi e l'accumulatore. Questi registri possono essere usati sia singolarmente che in coppia. Per il Gruppo 1, le tre coppie di registri di 16 bit sono:

- Il registro non specializzato di 16 bit formato dal registro B e dal registro C. Quando viene usato per indirizzare la memoria, il registro B corrisponde al byte HI dell'indirizzo e il registro C al byte LO.

- Il registro non specializzato di 16 bit formato dal registro D e dal registro E. Quando viene usato per indirizzare la memoria, il registro D corrisponde al byte HI dell'indirizzo e il registro E al byte LO.

- Il registro sia di indirizzo di memoria che non specializzato, a 16 bit, formato dal registro H e dal registro L. Quando viene usato per indirizzare la memoria, il registro H corrisponde al byte HI dell'indirizzo e il registro L al byte LO.

Per il Gruppo 2, i registri vengono accoppiati in modo analogo.

ACCUMULATORE

L'*accumulatore* è un registro a otto bit, interno al microprocessore Z-80, nel quale vengono posti i risultati della maggior parte delle operazioni aritmetiche e logiche. Nel caso del microprocessore Z-80, il registro accumulatore si trova all'interno del chip e può memorizzare un byte, cioé otto bit.

Fate molta attenzione a come potete modificare il contenuto dell'accumulatore: per esempio, potete sommare, sottrarre, confrontare i dati, incrementare, decrementare di 1 il suo contenuto. Potete scambiare il suo contenuto con una locazione di memoria o con dei dispositivi di ingresso/uscita. Potete ruotarne i bit sia verso destra che verso sinistra. Potete eseguire sul suo contenuto delle operazioni logiche, AND, OR e OR-esclusivo. Può darsi che a questo punto non abbiate capito alcune delle cose appena dette, ma abbiate pazienza ancora un pò: ne riparleremo tra poco.
Gli altri registri interni al micorprocessore Z-80 verranno trattati in un capitolo successivo.

ALCUNE ISTRUZIONI DELLO Z-80

Nel Capitolo 5, inizierete a fare esperimenti con i programmi del microcomputer. I programmi che eseguirete conterranno alcune istruzioni ad uno, due e tre byte, tra cui le seguenti:

00	NOP	Nessuna operazione
3C	INC A	Incrementa di 1 il contenuto dell'accumulatore
76	HALT	Alt del microcomputer
3E "dato"	LD A,dato	Carica il byte di dato, immediatamente successivo ("dato") nell'accumulatore
32 LO HI	LD (addr), A	Memorizza il contenuto dell'accumulatore nella locazione di memoria indirizzata dai due byte successivi (addr) di questa istruzione a tre byte
C3 LO HI	JP addr	Salta in modo incondizionato all'indirizzo di memoria indicato dai due byte successivi di questa istruzione a tre byte

Notate che questo elenco contiene istruzioni che hanno solo codici operativi ad un byte, cioè il primo byte di ogni istruzione. Il codice operativo esadecimale compare nella prima colonna accanto al codice mnemonico relativo.
Da questo momento in avanti, tutti i codici operativi, i byte di dati, i byte di indirizzo di memoria o i byte di spiazzamento saranno scritti in codice esadecimale, e tutti i codici mnemonici contenenti indirizzi o dati in codice esadecimale presenteranno il valore esadecimale seguito da "H".

Queste due istruzioni, illustrano un'importante notazione convenzionale:

```
LD (addr),A
JP addr
```

Nell'istruzione LD, "addr" è scritto fra parentesi, mentre le parentesi mancano nell'istruzione JP. Un indirizzo a sedici bit chiuso fra parentesi rappresenta il byte di dati che risiede alla locazione addr. Per esempio, l'istruzione

```
LD (0001H),A
```

viene eseguita ponendo il byte di dati presente nell'accumulatore, nell'indirizzo specificato da 0001. Le "( )" equivalgono a "l'indirizzo specificato da". L'indirizzo "addr" nell'istruzione JP di riferisce all'indirizzo dell'istruzione che deve essere successivamente eseguita dal computer. In questo caso viene effettuato un trasferimento del controllo del programma, e non un trasferimento di un dato come si verifica invece per l'istruzione LD. Quindi, un indirizzo a sedici bit che compare senza parentesi è un riferimento alla locazione stessa, mentre la presenza della parentesi indica che il riferimento è al *contenuto* di quella locazione. Si tratta di una sottile distinzione, che in seguito vi diventerà più naturale.

NOMENCLATURA DEI BYTE DI ISTRUZIONE

La letteratura della Intel Corporation, che illustra i codici mnemonici del microcomputer 8080, adotta le abbreviazioni e simboli seguenti per indicare il primo, il secondo e il terzo byte delle istruzioni a più byte:

< B1 > Primo byte di un'istruzione
< B2 > Secondo byte di un'istruzione
< B3 > Terzo byte di un'istruzione

Useremo questa notazione per facilitare la descrizione dei codici mnemonici dello Z-80. Per esempio, le istruzioni a tre byte JP e LD si possono scrivere nel modo seguente:

JP < B3 > < B2 >
LD (<B3><B2>),A

Analogamente, l'istruzione a due byte LD può essere scritta:

LD A,< B2 >

NESSUNA OPERAZIONE: NOP

L'istruzione più semplice dello Z-80 è l'istruzione NOP (Nessuna operazione) che ha il codice istruzione 00.

0 0 0 0 0 0 0 0

Non viene eseguita alcuna operazione. Potete usare questa istruzione in qualunque momento vogliate procurarvi dello spazio nel vostro programma per potervi aggiungere più avanti dei byte di istruzione. Imparerete che il vostro computer opera a circa 2,5 MHz, cioè 2,5 milioni di stati al secondo. *Tutte le istruzioni del microcomputer impiegano del tempo per essere eseguite.* Anche se non viene eseguita nessuna operazione, cioè la condizione dei registri e della memoria resta invariata, ciononostante l'istruzione NOP richiede 4 stati, o un totale di 1,6 microsecondi, per la sua esecuzione. Il tempo di esecuzione dipende dalla velocità del microcomputer. Se il microcomputer operasse a 4 MHz, 4.000.000 stati al secondo, l'istruzione NOP richiederebbe un tempo di esecuzione di 1 microsecondo. I microprocessori Z-80 A sono in grado di operare a 4 MHz, con alcuni chip speciali selezionati che operano ad una velocità di poco superiore.
In un paragrafo successivo, spiegheremo con precisione il significato della parola "stato".

ALT: HALT

Un'altra semplice istruzione dello Z-80 è l'istruzione di arresto, HALT, che ha il codice operativo 76,

0 1 1 1 0 1 1 0

Non appena questa istruzione viene eseguita, il microcomputer si ferma. Viene spesso usata per permettere al microcomputer di "aspettare" un'interruzione proveniente da un dispositivo esterno. In un computer, un'*interruzione* è una "sosta" nel flusso normale di una routine, tale che il flusso può riprendere più tardi da quello stesso punto. L'istruzione di HALT richiede 7 stati, per un tempo totale di esecuzione di 2,8 microsecondi.

INCREMENTA L'ACCUMULATORE: INC A

Il termine "incrementare" può essere così definito:

Incrementare	Aumentare il valore di una parola binaria. Di solito, aumentare il valore di 1.

L'istruzione di incremento, INC A, che ha un codice operativo 3C,

0 0 1 1 1 1 0 0

aumenta il contenuto del registro accumulatore di 1. L'istruzione INC A richiede 5 stati, o un tempo tottale di esecuzione di 2 microsecondi.

CARICA L'ACCUMULATORE IN MODO IMMEDIATO: LD A, data

Immediato si riferisce al fatto che il byte di dati è contenuto all'interno dell'istruzione a più byte. Nelle istruzioni IMMEDIATE, uno o due byte di dati a otto bit vengono acquisiti tramite un'istruzione a più byte che contiene il o i byte di dati nella forma di byte < B2 > o < B3 > ed eventualmente < B3 > o < B4 >. Questo fa parte del gergo dei microcomputer, che dovete imparare. L'istruzione "load immediate to accumulator" è un'istruzione a due byte che ha un codice operativo 3E, e i codici mnemonici LD A,<B2>.

0 0 1 1 1 1 1 0
byte di dati

Il secondo byte dell'istruzione è il byte di dati di otto bit che deve essere caricato nel registro accumulatore. L'intera istruzione a due byte richiede 7 stati, o 2,8 microsecondi, per la sua esecuzione.

Vedrete che questo è un sistema molto conveniente per alterare il contenuto dell'accumulatore. E' un'istruzione molto usata.

CARICA L'ACCUMULATORE IN MODO DIRETTO: LD (addr), A

Questa istruzione, LD (<B3><B2>), A è un'istruzione a tre byte che ha un codice operativo 32. Vi permette di porre il contenuto dell'accumulatore direttamente in una locazione di memoria, M, indicata dal secondo e dal terzo byte d'indirizzo dell'istruzione. Il secondo byte è il byte d'indirizzo LO e il terzo è il byte d'indirizzo HI.

0 0 1 1 0 0 1 0
byte d'indirizzo LO
byte d'indirizzo HI

Una volta eseguita l'istruzione, il programma non cambia il contenuto dell'accumulatore; copia soltanto il byte dell'accumulatore nella locazione di memoria indicata. Questa istruzione viene eseguita in 13 stati, o 5,2 microsecondi per un microcomputer a 2,5 MHz.

SALTO INCONDIZIONATO: JP addr

L'istruzione di salto incondizionato, JP <B3> <B2>, è un'istruzione a tre byte che ha un codice operativo C3. Il secondo byte dell'istruzione è il byte d'indirizzo di memoria LO, è il terzo byte è il byte d.indirizzo di memoria HI,

1 1 0 0 0 0 1 1
byte d'indirizzo LO
byte d'indirizzo HI

Una volta eseguita l'istruzione, il programma salta all'indirizzo di memoria a 16 bit dato dai byte d'indirizzo HI e LO. Questo tipo di istruzione è chiamata *Istruzione di Branch* (salto o collegamento). Essa vi permette di interrompere la sequenza normale di esecuzione del programma e di saltare in qualche altro punto della memoria, punto nel quale riprendete l'esecuzione del programma. Le istruzioni di salto sono molto utili. Esse vi permettono di scrivere *LOOP* di programma, gruppi di istruzioni cioé che vengono eseguiti in modo ripetitivo. In questo modo, siete in grado di ridurre sostanzialmente la complessità del programma. L'istruzione di salto incondizionato richiede 10 stati, o 4 microseconoi di tempo, per l'esecuzione dell'intera istruzione a tre byte.

DOMANDE RIEPILOGATIVE

1. Qual è la differenza fra un codice operativo, un byte di dati, un codice di dispositivo, un byte di indirizzo e un byte di spiazzamento?

2. Trovate il codice mnemonico dei seguenti codici di operazione binari a 8 bit:

   a. 01110110
   b. 00111110
   c. 11000011
   d. 00110010
   e. 00111100
   f. 00000000

3. Scrivete il codice operativo esadecimale a due digit per le seguenti istruzioni dello Z-80:

   a. HALT
   b. JP < B2>< B3 >
   c. LD (< B2 >< B3 >), A
   d. NOP
   e. INC A
   f. LD A,< B2>

4. In un'istruzione multi-byte dello Z-80, il codice operativo può essere il secondo, il terzo o il quarto byte dell'istruzione?

5. Per un microcomputer che opera a 2,5 MHz, quanto tempo occorre per eseguire le seguenti istruzioni:

   a. JP
   b. LD A,< B2 >
   c. INC A

6. Fate un elenco dei sei registri non specializzati e dei sei registri specializzati del microprocessore Z-80. Indicate quali dei registri non specializzati vengono usati a coppie. Che cosa significa ARS?

RISPOSTE

1. Il codice operativo è il codice a 8 bit che indica l'operazione che il microprocessore Z-80 eseguirà. Un byte di dati è un numero binario a 8 bit che lo Z-80 userà in un'operazione aritmetica o logica o che immagazzinerà in memoria. Un codice di dispositivo è l'identificazione del dispositivo specifico di ingresso/uscita con il quale l'accumulatore di un microprocessore Z-80 scambierà otto bit di informazione. Un byte d'indirizzo è costituito dagli otto bit più significativi, o meno significativi che formano la parola dell'indirizzo di memoria a 16 bit dello Z-80. Un byte di spiazzamento è un numero a 8 bit con segno, complemento a due, che viene usato per l'indirizzamento indicizzato.

2. a. HALT
   b. LD A,<B2>
   c. JP <B2><B3>
   d. LD (<B2><B3>), A
   e. INC A
   f. NOP

3. a. 76
   b. C3
   c. 32
   d. 00
   e. 3C
   f. 3E

4. In un'istruzione multi-byte, il primo, il secondo, e il quarto byte possono essere codici operativi. Il primo byte, che è sempre un codice operativo, determina il significato del secondo byte. Se il secondo byte è un codice operativo, esso determina il significato dei byte rimanenti, se ce ne sono.

5. a. 4,0 microsecondi
   b. 2,8 microsecondi
   c. 2,0 microsecondi

6. I sei registri non specializzati sono B, C, D, E, H e L.
   I sei registri specializzati sono SP, PC, IX, IY, I e R.
   I registri non specializzati vengono accoppiati in tre registri a 16 bit in questo modo: BC, DE e HL.
   ARS significa "alternate register set "(set di registri alternativi), cioé un secondo insieme o gruppo di registri non specializzati, di flag e di accumulatore: A', B', C', D', E', F', H', e L'.

CAPITOLO 4

IL NANOCOMPUTER NBZ80 E IL NANOCOMPUTER SUPER NBZ80S

INTRODUZIONE

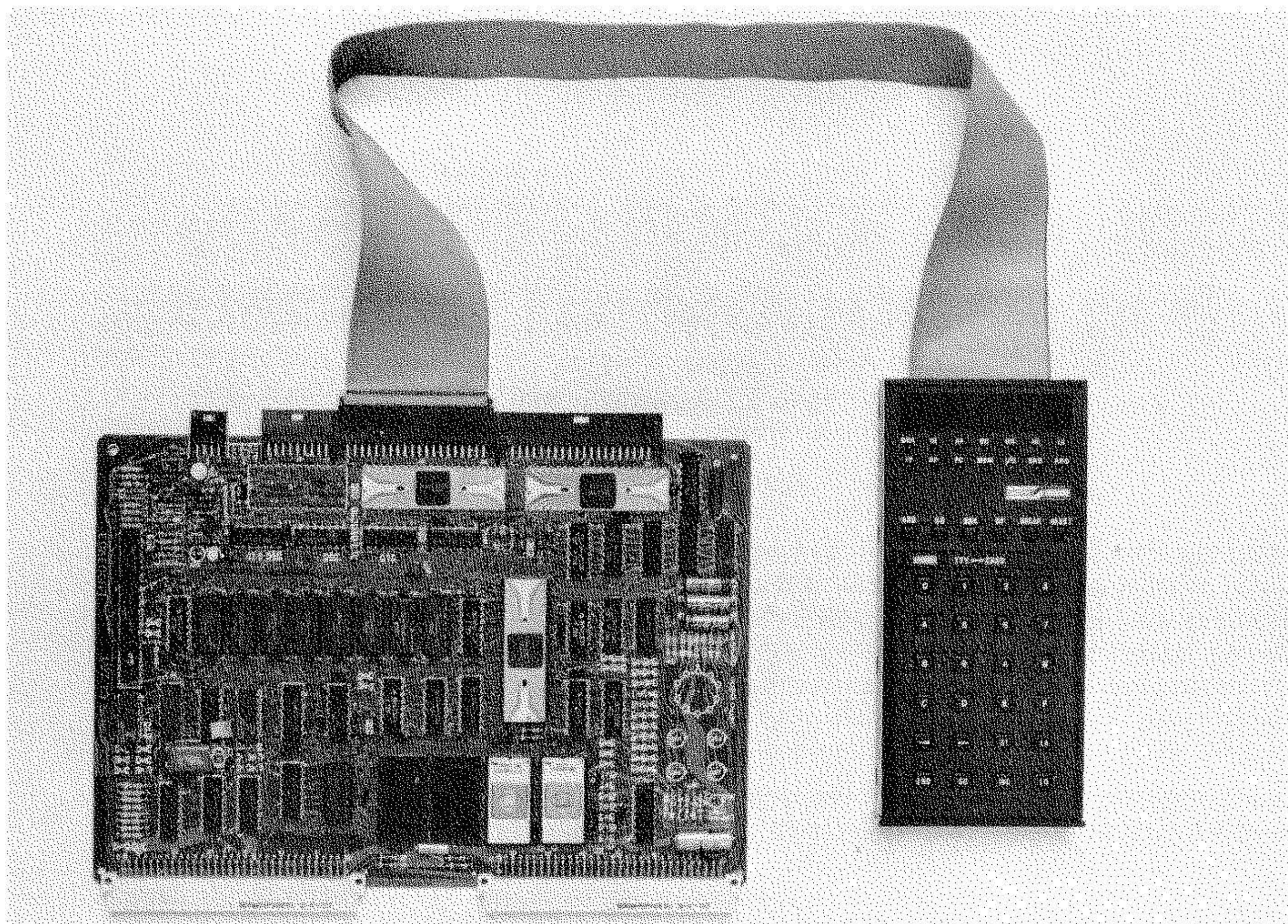
Nei capitoli che seguono, eseguirete degli esperimenti che illustrano alcuni importanti concetti relativi alla programmazione e all'interfacciamento dei microcomputer. Per eseguire questi esperimenti, userete il Nanocomputer (un microcomputer basato sullo Z-80), ed inoltre alcuni circuiti integrati, alcune basette (breadboard) per collegamenti senza saldature, fili, ad altri componenti circuitali. In questo capitolo verranno discussi alcuni di quesri argomenti, in modo che sarete pronti ad usare correttamente tali dispositivi quando eseguirete gli esperimenti.

Il Nanocomputer Z-80 è prodotto dalla SGS-ATES Componenti Elettronici S.p.A. con sede in Agrate Brianza Via C. Olivetti, 2 - 20041 Italia.

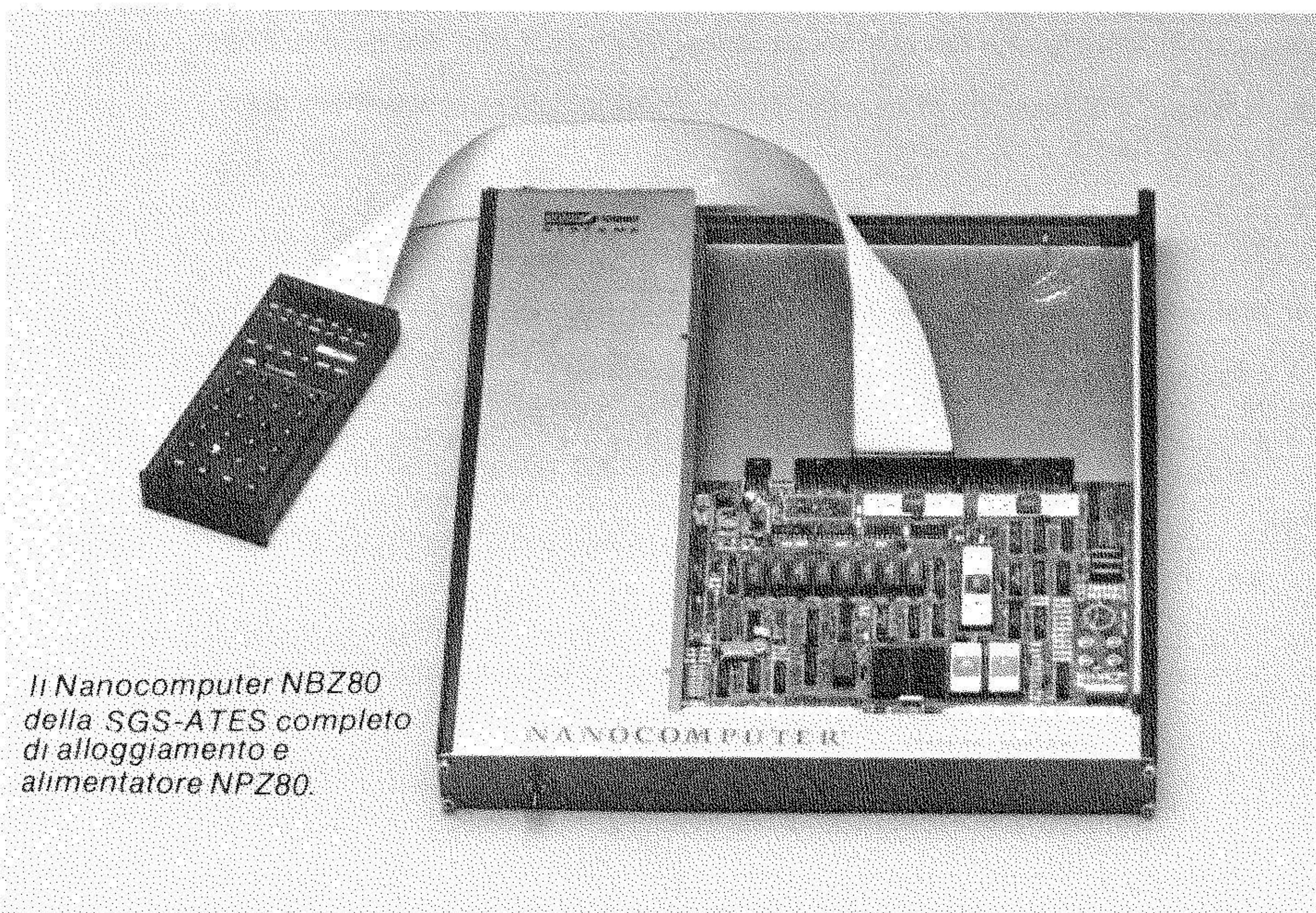
OBIETTIVI

Alla fine di questo capitolo, sarete in grado di:

- Elencare le esigenze del Nanocomputer per quanto riguarda l'alimentazione
- Stabilire la funzione di ognuno dei trenta tasti della tastiera del Nanocomputer
- Spiegare il significato di ognuno dei quattordici LED della tastiera del Nanocomputer
- Identificare e spiegare il significato degli otto display a sette segmenti presenti sulla tastiera del Nanocomputer
- Stabilire la frequenza e la durata di uno stato del Nanocomputer
- Caricare ed eseguire un semplice programma
- Stabilire quali terminali senza saldatura sono elettricamente collegati insieme su di un breadboard
- Spiegare la differenza fra memoria a lettura/scrittura e memoria programmabile a sola lettura
- Dare la posizione della memoria a lettura/scrittura e della memoria a sola lettura nel Nanocomputer e indicare l'indirizzo di partenza del suo sistema operativo contenuto nella memoria a sola lettura.



Il Nanocomputer NBZ80 della SGS-ATES.



Il Nanocomputer NBZ80 della SGS-ATES completo di alloggiamento e alimentatore NPZ80.

IL NANOCOMPUTER

FINALITA'

Il Nanocomputer è un piccolo microcomputer basato sullo Z-80 con 4K di memoria a lettura/scrittura e 2K di memoria PROM/ROM. Esistono due versioni del Nanocomputer:

(1) NBZ80 - un microcomputer a scheda aperta con un terminale di ingresso dati e display, cui di solito si fa riferimento come tastiera del Nanocomputer

(2) NBZ80-S - una scheda NBZ80 racchiusa in un involucro, comprensivo di un breadboard per gli esperimenti e di alimentazione. La "S" indica super.

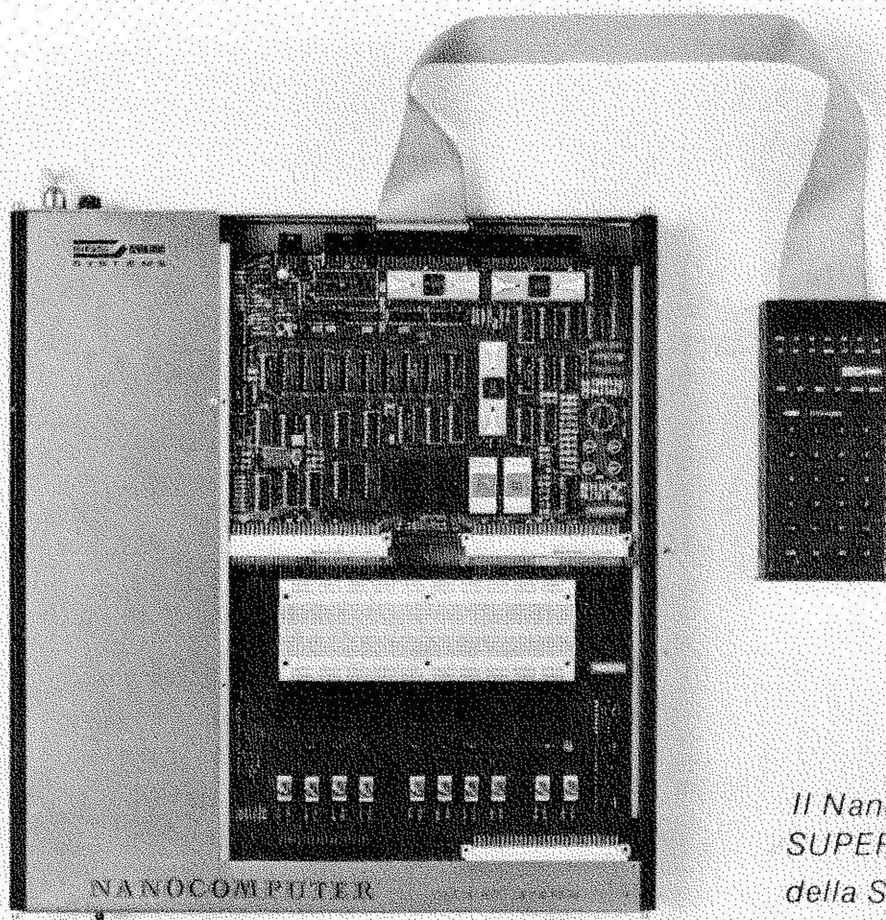
Entrambi i Nanocomputer sono progettati a scopo educativo, per insegnare a interfacciare e programmare la CPU Z-80.
Entrambi possono essere usati come computer a sè stanti, oppure possono essere integrati in sistemi complessi costituiti da altri microcomputer e/o da grossi computer.
Nei prossimi capitoli vi farete un'esperienza nello sviluppo del software dei microprocessor, mentre i capitoli successivi, che trattano l'interfacciamento, vi presenteranno degli esperimenti pratici da eseguire con il Nanocomputer.

Eseguirete tre tipi di esperimenti:

(1) Esperimenti che richiedono la programmazione del Nanocomputer ed usano solo il Nanocomputer NBZ80 e un alimentatore NPZ80

(2) Esperimenti che implicano la costruzione di circuiti digitali e richiedono un breadboard, alimentazione, ed alcuni componenti digitali

(3) Esperimenti che implicano sia la programmazione che la costruzione di circuiti digitali di interfaccia, e richiedono il Nanocomputer Super (NBZ80-S).



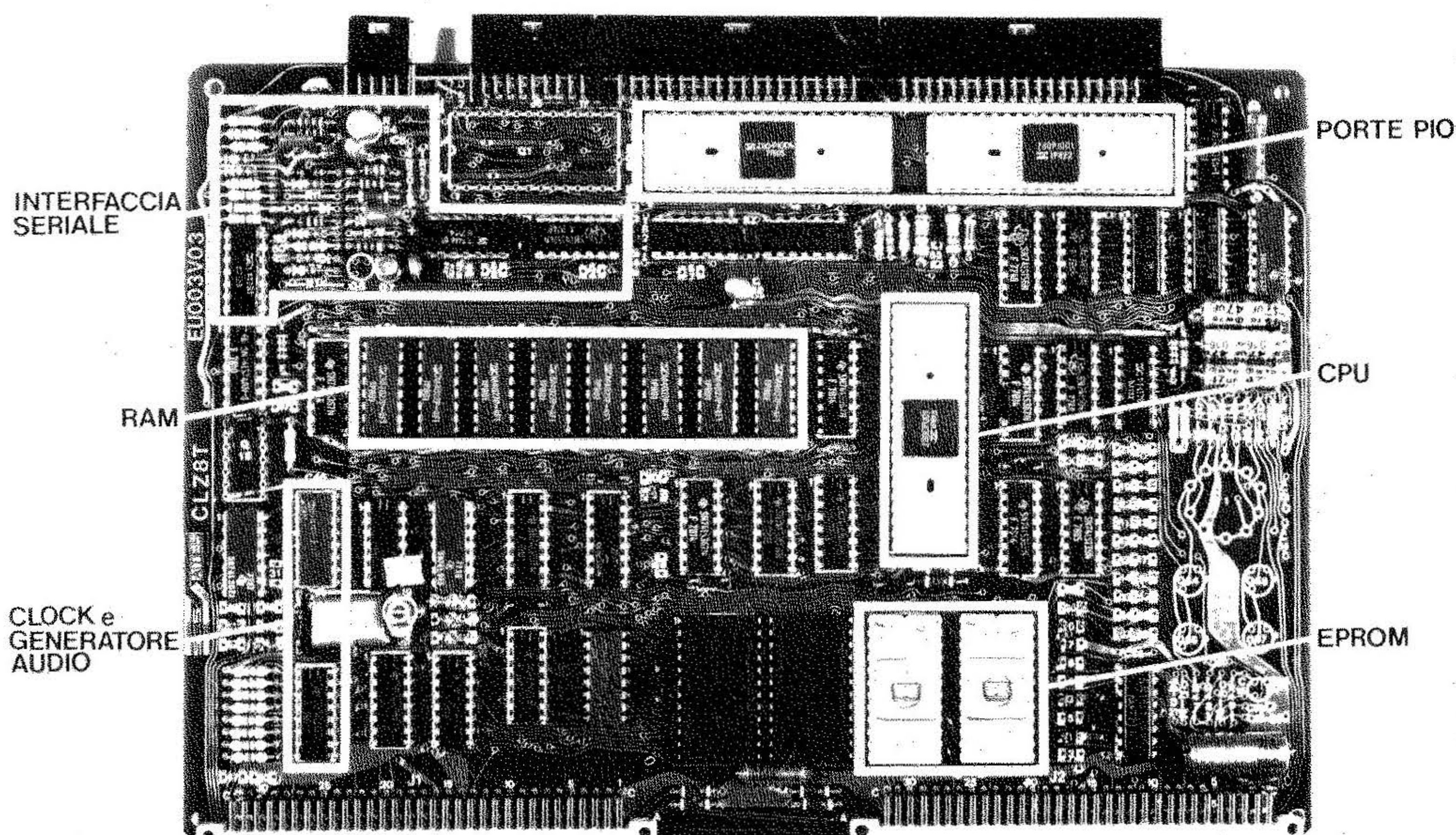
Il Nanocomputer SUPER NBZ80-S della SGS-ATES.

DESCRIZIONE

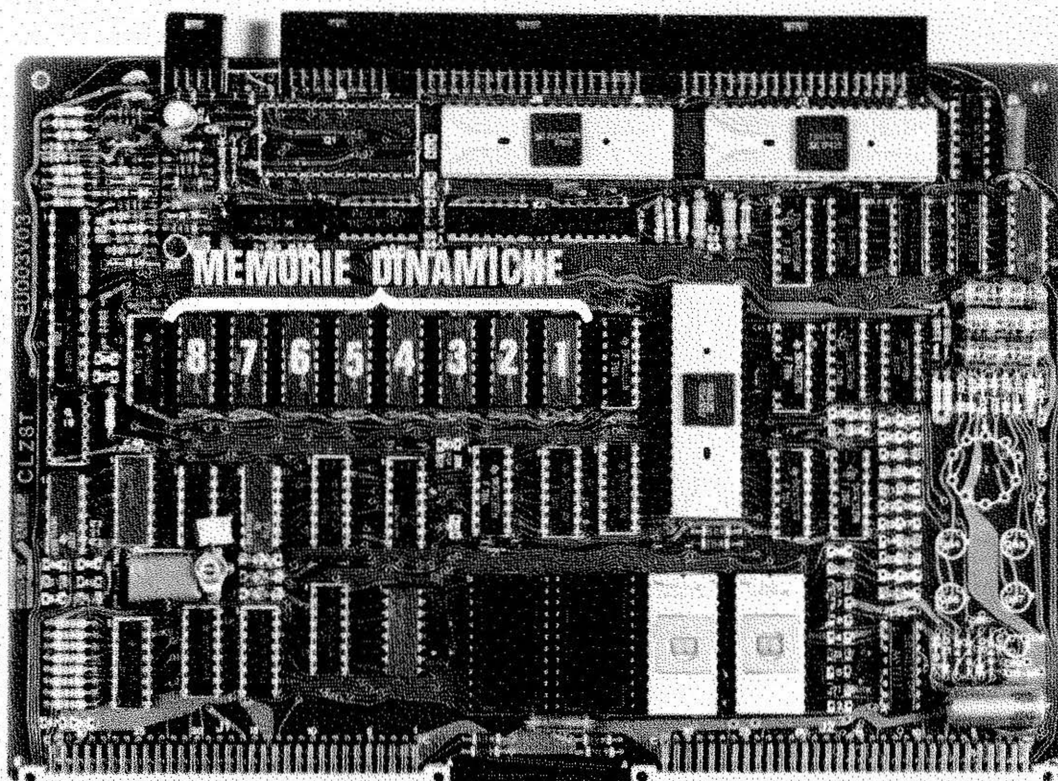
Il Nanocomputer è un microcomputer basato sullo Z-80 e costituito da una sola scheda che contiene la CPU Z-80 SGS-ATES, 2 chip PIO, la memoria ed una tastiera di trenta tasti con display per l'ingresso dei dati. La tastiera permette all'utente di caricare i programmi nella memoria del microcomputer, di selezionare locazioni di memoria specifiche per la lettura e la scrittura in memoria; di eseguire programmi alla massima velocità, e passo-passo, di resettare il microcomputer allo stato iniziale, e di eseguire molte altre funzioni.
Vi descriveremo tutte queste funzioni in dettaglio.
Nelle pagine seguenti potete vedere alcune fotografie del Nanocomputer.
Sulla piastra del circuito stampato di un Nanocomputer potete riconoscere i seguenti blocchi funzionali:

- CPU
- Memoria RAM (lettura/scrittura)
- Memoria ROM o EPROM
- 2 Porte parallele di I/O (2 circuiti integrati PIO)
- 2 Porte seriali di I/O (interfaccia per terminali seriali e cassette audio)
- Bus driver
- Generatore di clock e Baud Rate

Non è necessario che questa suddivisione in blocchi funzionali vi sia chiara sin dal principio. Inizialmente, dovrete imparare ad usare la tastiera e a leggere ed interpretare l'uscita. Man mano che diventerete esperti nella programmazione dei microcomputer incomincerete a capire nei dettagli i diversi circuiti costituenti il Nanocomputer.



La scheda base del Nanocomputer NBZ80 con evidenziata la partizione in blocchi funzionali.



La scheda base del Nanocomputer NBZ80 con evidenziata la posizione dei dispositivi di memoria.

ALIMENTAZIONE

Il Nanocomputer richiede le seguenti alimenazioni:

+ 5 V ± 5% a 800 mA
− 5 V ± 5% a 200 mA
+ 12 V ± 10% a 100 mA
− 12 V ± 10% a 100 mA

Questi tipi di alimentazione sono compresi nella versione NBZ80-S. Per l'NBZ80, la SGS-ATES fornisce un'alimentatore adatto, (NPZ80) che si può richiedere a parte.

LA TASTIERA DEL NANOCOMPUTER

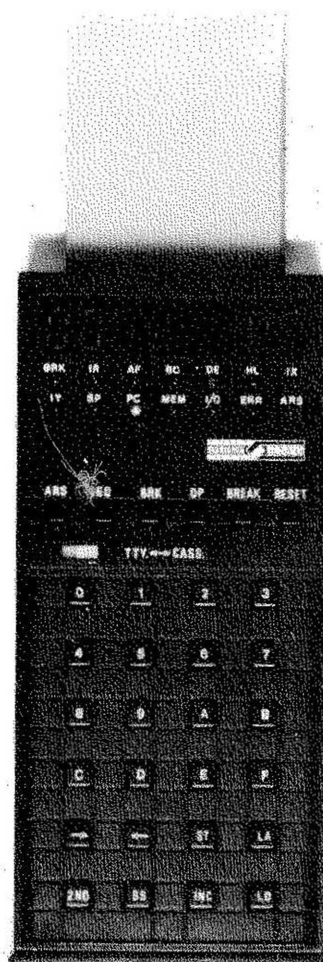
La tastiera del Nanocomputer è collegata al circuito stampato tramite un flat-cable (cavo piatto) da 40 linee. Una foto della tastiera del Nanocomputer è presentata a pag. 4-6. Quanto segue è una descrizione completa di ciascun tasto, descrizione cui potrete sempre far riferimento. Non cercate di imparare a memoria queste descrizioni, ma piuttosto prendete diretta visione del materiale e verificate sperimentalmente la funzionalità dei singoli tasti tramite gli esperimenti suggeriti alla fine di questo e dei successivi capitoli.

Da 0 ad F:

Questi tasti permettono di inserire un digit esadecimale all'estrema destra del display dei dati a quattro digit.
Ad ogni inserzione di un digit sulla destra, gli altri tre subiscono uno shift a sinistra, con la conseguente perdita del digit più a sinistra.

Freccia a sinistra (←) e freccia a destra (→):

Questi due tasti sono utilizzati per selezionare (accendere) uno dei quattordici led posti immediatamente al di sotto dei display dei dati e degli indirizzi.



L'unità di controllo (tastiera + display) del Nanocomputer NBZ80 della SGS-ATES.

Tutti i led, ad esclusione di ARS, BRK ed ERR, possono essere accesi muovendo la luce di selezione a sinistra oppure a destra.
Notate che, tenendo premuto uno di questi due tasti, la luce di selezione si sposta automaticamente.
Infine, quando l'indicazione luminosa raggiunge l'ultimo led, in entrambe le direzioni, si inizia un nuovo ciclo.

Chiariamo ora il significato delle visualizzazioni che si ottengono scegliendo differenti posizioni di selezione.
In corrispondenza delle posizioni IR, AF, BC, DE, HL, sul display dei dati compaiono quattro digit esadecimali (i quattro digit di destra). Questi quattro digit rappresentano due byte di dati. I due digit di sinistra indicano rispettivamente i contenuti dei ragistri I, A, B, D e H, mentre i due digit di destra indicano rispettivamente i contenuti dei registri R, F, C, E e L, in funzione della posizione della luce di selezione. Per le posizioni IX, IY, SP, PC e MEM, i quattro digit esadecimali sul display degli indirizzi (address display) sulla sinistra, visualizzano il contenuto dei registri a 16 bit selezionati, mentre i due digit esadecimali nella zona dati del display (data display) danno il contenuto della locazione puntata dal registro selezionato. Quando la luce di selezione è posta sul led I/O, il display degli indirizzi presenta un codice dispositivo ad un byte ed il display dei dati visualizza il contenuto di quella porta.

ST: ST è una abbreviazione per Store. La sua funzionalità dipende dalla posizione dell'indicatore luminoso. Se l'indicatore luminoso è nella posizione IR, AF, BC, DE o HL, quando viene premuto questo tasto i digit esadecimali occupanti le posizioni di estrema destra (il byte di ordine più basso) del data display sono memorizzati (STORED) nei registri R, F, C, E ed L.
Se l'indicatore luminoso è nella posizione IX, IY, SP e PC, i quattro digit (due byte) presenti sul data display sono memorizzati nei registri a 16 bit IX, IY, SP o PC. Con l'indicatore su MEM, i due digit (un byte) esadecimali di destra, sono memorizzati all'indirizzo che compare sull'address display, con successivo autoincremento dell'indirizzo visualizzato, al fine di puntare alla successiva locazione di memoria.

Infine, se è acceso il led di I/O, i due digit (un byte) esadecimali di destra, sono posti in uscita alla porta di uscita selezionata dall'address display, che è successivamente incrementato di 1.

Si noti che la memorizzazione avviene solo se i digit sono introdotti nel data display da tastiera.

LA: LA è l'abbreviazione di Load Address. Quando il led BRK è spento (cioé quando il Nanocomputer non è in Breakpoint mode, vedi oltre), il tasto LA può essere usato solo quando l'indicatore luminoso è nella posizione MEM o I/O.
In qualsiasi altra posizione dell'indicatore, l'utilizzo di LA determinerà l'accensione del led verde ERR di errore, da cui l'indicazione di una operazione non valida.
Quando il led di BRK è acceso, il tasto LA ha un diverso utilizzo. Una più ampia descrizione dell'utilizzo di LA in Breakpoint mode è data nel paragrafo relativo al tasto BRK.

Con il Nanocomputer non il in Breakpoint mode, se l'indicatore luminoso è in posizione MEM, LA esegue le seguenti operazioni:

a) i quattro digit esadecimali precedentemente caricati, attualmente visualizzati nel data display, vanno nell'address display (a sinistra), e

b) i contenuti della locazione di memoria puntata dall'indirizzo presente nell'address display, sono visualizzati sul data display.

2ND: 2ND si riferisce al secondo byte, o byte di ordine più alto, delle coppie di registri IR, AF, BC, DE e HL, si riferisce cioé ai registri I, A, B, D ed H.
Per memorizzare un byte (due digit esadecimali) in questi registri, la procedura è:
Passo 1. Posizionate l'indicatore luminoso in corrispondenza della coppia di registri desiderata (IR, AF, BC, DE, HL)
Passo 2. Caricate due digit esadecimali nel data display
Passo 3. Premete il tasto 2ND
Passo 4. Premete il tasto ST

Come risultato, i contenuti della coppia di registri sono nuovamente visualizzati nel data display, con il byte di ordine più alto modificato come desiderato. Il byte di ordine più basso è inalterato. Notate che se vengono inseriti più di due digit esadecimali, solo gli ultimi due vengono caricati nel byte di ordine più alto. Il tasto 2ND non ha effetto quando l'indicatore luminoso è posizionato su IX, IY, SP, PC, MEM o I/O, in quanto questi "registri" non sono coppie di registri ad 8 bit, come lo sono IR, AF, BC, DE,HL.

SS: SS indica "Single Step". Questa è una caratteristica molto utile del Nanocomputer, permettendo l'esecuzione di programmi un passo alla volta.
Dopo ogni passo, i contenuti dei vari registri possono essere esaminati, da cui la possibilità di utilizzo di un interessante supporto per collaudare un programma.
Utilizzeremo la funzione Single Step per illustrare alcuni dettagli della operatività del microprocessore Z-80, alcuni dei quali possono essere verificati solo quando lo Z-80 opera a velocità "rallentata".

E' interessante notare che il Single Step, sul Nanocomputer, è realizzato da software, mentre in genere viene realizzando circuiti particolari hardware.

Parleremo in seguito di questa interessante soluzione
Dapprima si carica un programma, si pone lo Start address (indirizzo di partenza) nel registro PC e poi si preme SS. Ogni qualvolta SS è premuto, viene eseguita una istruzione. Tra successive pressioni di SS, l'utente può posizionare l'indicatore luminoso in corrispondenza di qualunque posizione, al fine di selezionare quello che si vuole visualizzato.
L'esecuzione passo-passo di un programma può anche iniziare dopo un Breakpoint (le informazioni sui Breakpoint saranno fornite in seguito).
Tenendo premuto il tasto SS, si determina una esecuzione sequenziale lenta del programma fino al rilascio del tasto stesso.

INC: INC è l'abbreviazione per Increment.

Questo tasto ha due funzioni. Primo, quando il Nanocomputer non è in Break point mode e la luce di selezione è nella posizione MEM o I/O, l'indirizzo della locazione di memoria o il codice dispositivo presente sull'address display è incrementato di uno, visualizzando quindi la successiva locazione di memoria ed il relativo contenuto. In qualsiasi altra posizione dell'indicatore luminoso, l'utilizzo del tasto INC determina l'accensione del led ERR, dando indicazione di operazione non permessa.
Il secondo utilizzo del tasto INC è in corrispondenza del Breakpoint mode del Nanocomputer. L'esatto utilizzo del tasto INC in Breakpoint mode sarà descritto in seguito nel paragrafo relativo al tasto BRK.

ARS: ARS indica Alternate Register Set. Questo tasto attua lo scambio tra i due set di registri A, B, C, D, E, F, H, L e A', B', C', D', E', F', H', L'.

Premendo il tasto ARS con l'indicatore luminoso in posizione AF, sul data display vengono visualizzati i registri A' ed F'. Analogo risultato si ottiene per le posizioni BC, DE e HL. Non vi sono modifiche del display in corrispondenza di altre posizioni dell'indicatore luminoso. Quando è visualizzato il set alternativo di registri, il led ARS è acceso. Notate come il tasto ARS accende e spegne alternativamente il led ARS.

GO: Questo tasto ha due funzioni. La prima consiste nell'iniziare o riprendere l'esecuzione di un programma del microcomputer. La seconda consiste nel rimuovere il Breakpoint, la cui descrizione verrà fatta nel paragrafo relativo al tasto BRK.
Per iniziare l'esecuzione di un programma, occorre specificare l'indirizzo di partenza del programma. Questa operazione può essere fatta in due modi:

1. Caricare il PC (Program Counter) con l'indirizzo di partenza e poi premere GO per lo start dell'esecuzione del programma.
2. Caricare l'indirizzo di partenza sul data display e premere GO.

In entrambi i casi, l'esecuzione del programma continuerà fino a quando non vengano incontrati una situazione di HALT o un breakpoint. Per riprendere l'esecuzione dopo un breakpoint, premere semplicemente GO un'altra volta.

BRK: BRK è una abbreviazione per Breakpoint. Questo tasto pone il Nanocomputer entro o fuori del Breakpoint Mode, rispettivamente led BRK acceso o spento. Premendo più volte BRK, il led BRK si accende e si spegne alternativamente.

Vediamo ora di definire cosa si intende per breakpoint e quale è il suo significato per il vostro Nanocomputer, quando si è nel Breakpoint Mode.
Un breakpoint è un arresto (break) nell'esecuzione di un programma.
Voi definite un breakpoint specificando una istruzione (cioé l'indirizzo del primo byte dell'istruzione) in corrispondenza della quale l'esecuzione deve arrestarsi. Potete così esaminare i contenuti dei registri e delle varie locazioni di memoria, prima di riprendere l'esecuzione del programma. L'esecuzione può riprendere in Single - Step, (utilizzando il tasto SS) oppure in mode continuo (premendo il tasto GO). Voi specificate un breakpoint, fornendo un indirizzo (due byte oppure quattro digit esadecimali).
Questo indirizzo deve puntare al primo byte, nel caso di una istruzione a più byte. E' importante tener presente che l'istruzione che inizia all'indirizzo di breakpoint NON è eseguita, quando si incontra il breakpoint.
Questa istruzione sarà eseguita solo alla ripresa dell'esecuzione del programma.
Potete definire fino a 8 indirizzi di breakpoint alla volta. Questi indirizzi sono numerati da 0 a 7. Questa numerazione serve solo come riferimento e non si riferisce all'ordine di inserimento dei breakpoint.

Ad esempio, il breakpoint 0 può verificarsi, nell'ambito dell'esecuzione di un programma, dopo il breakpoint 5. La sequenza di passi di seguito riportata descrive come definire dei breakpoint, utilizzando i tasti BRK, INC ed LA.

Passo 1. Premere il tasto BRK, per entrare nel Breakpoint Mode. Il Nanocomputer è ora pronto ad accettare la definizione dei vari breakpoint.

Passo 2. Sul data display dovrebbe apparire uno 0. Premendo il tasto INC, si incrementa questo numero fino a 7 e poi di nuovo a 0. Questo digit indica il numero corrente di breakpoint.

Passo 3. Per definire l'indirizzo di un breakpoint, visualizzate il numero di breakpoint scelto, impostate l'indirizzo (quattro digit) e premete il tasto LA. Il display dovrebbe dare ora le seguenti informazioni:
a) I primi quattro digit sono gli indirizzi introdotti.
b) C'è un digit spento seguito da tre digit.
c) Il primo digit del gruppo di tre digit è il numero del breakpoint.
d) Il secondo e il terzo digit del gruppo di tre digit costituiscono il contenuto dell'indirizzo del breakpoint, cioé il primo byte dell'istruzione in corrispondenza della quale si vuole il breakpoint.

Passo 4. Dei successivi breakpoint possono essere definiti INCrementando fino al numero di breakpoint desiderato, introducendo un indirizzo e premendo LA.

Qualsiasi breakpoint già impostato può essere modificato con la stessa procedura.

Passo 5. Premere ancora BRK per uscire dal Breakpoint Mode.

Per eliminare un breakpoint, bisogna entrare nel Breakpoint Mode, incrementare il numero di breakpoint fino ad arrivare a quello che si desidera eliminare, poi premere GO. Sul display dovrebbe apparire solo il numero del Breakpoint.

LD e DP: Se al vostro Nanocomputer è collegato un registratore a cassetta, potete usare la cassetta come supporto per memoria di massa, cioè potete trasferire sul nastro della cassetta programmi o dati che sono memorizzati nella memoria del Nanocomputer (questo con il tasto DP); oppure potete caricare da nastro programmi o dati precedentemente salvati su nastro (questo con il tasto LD).

Dato che il contenuto della memoria lettura scrittura viene sempre distrutto quando si toglie l'alimentazione al Nanocomputer, un supporto fisico di memoria di massa come la cassetta magnetica mette a disposizione un comodo ausilio per il ristabilimento della memoria ad uno stato voluto, dopo che è stata ridata l'alimentazione.
In effetti, nel realizzare gli esperimenti di programmazione in questo capitolo e nei successivi, vi raccomandiamo caldamente di trasferire su cassetta i programmi più lunghi. In tal modo, nel caso in cui si rendesse necessario ricaricare un dato programma, non avete bisogno di reintrodurlo byte per byte da tastiera, ma è sufficiente utilizzare il tasto LD.
E' importante ricordare che i tasti LD e DP possono essere utilizzati anche con dispositivi diversi dalle cassette magnetiche. Un dispositivo seriale ASCII può inviare o ricevere dati al/dal Nanocomputer. Il che vuol dire che i comandi DP e LD possono essere utilizzati per inviare dati a perforatori di nastro di carta, stampanti, etc., e per ricevere dati da perforatori di nastro, schermi CRT, etc.
Occorre, a questo proposito informare il Nanocomputer, tramite l'interruttore TTY/CASS presente sulla tastiera, se si ha a che fare con una teletype seriale (TTY) oppure con una cassetta audio (CASS).

Verrà discusso in modo specifico come utilizzare il tasto TTY/CASS in seguito, nell'ambito di una discussione più dettagliata sull'I/O relativo ad una cassetta audio.
La SGS-ATES fornisce, per l'interfacciamento con il Nanocomputer, il registratore a cassette audio RCZ80.
Può comunque essere utilizzata qualsiasi cassetta standard.
Quanto segue si applica in modo specifico alla RCZ80, tuttavia, con piccole modifiche, può essere applicato pari pari a dispositivi di diversa fabbricazione. In questo ultimo caso, fate riferimento al manuale operativo.
Come operazioni preliminari all'utilizzo del registratore a cassette occorre fare quanto segue:

- selezionare la tensione di alimentazione (110/120 o 220/240 V ca); utilizzare a questo proposito il selettore presente sulla parte posteriore del registratore
- collegarsi alla rete
- posizionare il controllo di volume a volume massimo (10)
- con il Nanocomputer spento, collegare il cavo della cassetta: il connettore circolare a 7 pin si inserisce nella sede corrispondente sul registratore, ed il connettore piatto ad 8 pin si inserisce nel connettore J3 della scheda CL Z80 (parte superiore di sinistra)
- posizionare l'interruttore TTY/CASS sulla tastiera del Nanocomputer sulla posizione CASS.

L'OPERAZIONE DP

Il tasto DP viene premuto quando si vuole dar inizio ad una operazione di scrittura su cassetta. La procedura per la registrazione dei contenuti di un blocco continuo di memoria del Nanocomputer (RAM, ROM, EPROM) è la seguente:

1. Accendere il Nanocomputer
2. Posizionare il nastro alla posizione iniziale di registrazione (si utilizzino i tasti di avanzamento rapido (>>) e di riavvolgimento (<<).
3. Porre l'indicatore luminoso di selezione sul Nanocomputer alla posizione MEM.
4. Inserite servendovi della tastiera del Nanocomputer l'indirizzo di partenza del blocco di memoria che si vuole trasferire (fino ad un massimo di quattro digit esadecimali di indirizzo) e premere LA.
5. Inserite, sempre da tastiera, la lunghezza del blocco che si vuole trasferire. Il display della tastiera dovrebbe mostrare l'indirizzo di partenza alla sinistra e la lunghezza del blocco a destra.
6. Premere il tasto DP sulla tastiera del Nanocomputer. Il display verrà spento.
7. Verificare di nuovo che il deviatore TTY/CASS sia nella posizione CASS e premere sul registratore il tasto di registrazione e il tasto di avanzamento (>) contemporaneamente. Il nastro ancora non comincia a muoversi, mentre si udirà un suono continuo.
8. Premere il tasto GO sulla tastiera del Nanocomputer. Il nastro comincerà a muoversi. Dopo circa 20" inizierà l'operazione di registrazione.

9. Una volta terminata la scrittura sulla cassetta, il registratore si arresta automaticamente. Abbiamo avuto modo di verificare che sono necessari circa 20-25 secondi per scrivere 256 byte, quindi non preoccupatevi se l'operazione di scrittura sembra essere ben più lunga di quanto non vi foste aspettati. Dopo l'arresto della cassetta, sbloccate i tasti di avanzamento e di registrazione.

10. Premere un tasto qualsiasi sulla tastiera del Nanocomputer per ritornare alla operatività normale. A questo punto è possibile riavvolgere il nastro e toglierlo dal registratore.

L'OPERAZIONE LD

Il tasto LD è utilizzato per caricare in memoria programmi e/o dati precedentemente registrati su cassetta. L'indirizzo di partenza ed il numero di byte sono stati memorizzati su nastro unitamente al programma, quindi non è necessario specificarli di nuovo.

Di seguito è data la sequenza di operazioni da eseguirsi per attivare l'operazione di trasferimento dati dalla cassetta alla memoria.

1. Accendere il Nanocomputer

2. Posizionare il nastro alla posizione iniziale di lettura (utilizzando l'avanzamento rapido (>>) oppure il riavvolgimento (<<)

3. Verificare di nuovo che l'interruttore TTY/CASS sulla tastiera del Nanocomputer sia nella posizione CASS.

4. Premere il tasto LD. Il display del Nanocomputer a questo punto è completamente spento. L'operazione di caricamento è così iniziata.

5. Premere il tasto di avanzamento (>); si dovrebbe sentire a questo punto un suono molto acuto, dovuto alla lettura dei dati registrati.

6. Il registratore si arresterà al termine dell'operazione di lettura e verrà emesso un suono inconfondibile.

7. Una volta che il registratore si è arrestato, sbloccate il tasto di avanzamento.

8. Se sulla tastiera del Nanocomputer è accesa l'indicazione luminosa ERR, vuol dire che si è verificato un errore nella lettura dei dati. In questo caso si effettui una nuova operazione di lettura. Se il led ERR si accende ancora, allora vuol dire che il nastro è difettoso o che è stato scritto in modo errato.

9. Se il caricamento è stato corretto, il led ERR è spento.

10. Premere un tasto qualsiasi sulla tastiera del Nanocomputer per ritornare all'operatività normale.

Nota 1:

Con il Nanocomputer è anche possibile leggere cassette che contengono programmi generati dal monitor MO-Z o dall'assemblatore ASS-Z del sistema CLZ80 della SGS-ATES.
Per effettuare tale operazione è necessario scrivere nella locazione INMODE (il cui indirizzo assoluto è riportato della appendice IV) un valore diverso da 00 esadecimale. La procedura di caricamento (LD) sarà poi quella usuale.
Si noti che i programmi generati dal Nanocomputer non possono invece essere letti dal monitor MO-Z nè dall'assemblatore ASS-Z del sistema CLZ80.

Nota 2:

La frequenza di trasmissione/ricezione seriale (baud rate) è programmabile dall'utente. All'accensione o dopo un RESET essa è automaticamente fissata a 600 baud. Per programmare differenti velocità occorre scrivere un opportuno valore nelle locazioni di indirizzo (espresso in modo simbolico) BAUDRT e BAUDRT + 1, secondo la seguente tabella (relativa ai valori di baud rate più comuni):

BAUDRT (I° BYTE)	BAUDRT + 1 (II° BYTE)	BAUD RATE
9A	00	600
35	01	300
55	03	110

Nota 3:

Il formato dei byte registrati su nastro è il seguente:

- Ciascun byte di memoria è tradotto in due caratteri ASCII, un carattere ASCII per ciascun digit esadecimale. Ad esempio il byte di memoria 00101010 è tradotto in un 2 (ASCII) ed in una A (ASCII).

- I byte di memoria sono raggruppati in record.

- Ciascun record ha il seguente formato:

Caratteri	Contenuti
1	Ritorno di carrello
2	Line feed (avanzamento carta)
3	Due punti (:)
4-5	Lunghezza del record
6-7	Indirizzo di memoria di partenza del record, byte HI
8-9	Indirizzo di memoria di partenza del record, byte LO
10-11	non utilizzato
12-N	bytes di dati, due caratteri ASCII per byte (numero dipendente dal campo lunghezza del record, caratteri 4 e 5)
(N + 1)-(N + 2)	Checksum (somma di controllo)

BREAK: Il tasto BREAK può essere considerato come pulsante di emergenza. La pressione del tasto BREAK determina un interrupt non mascherabile che, a sua volta, causa l'immediata interruzione del programma in corso. Il controllo è restituito al sistema operativo del Nanocomputer, con la luce di selezione posizionata in corrispondenza di PC. La parte indirizzo del display punta all'ultima istruzione eseguita, e gli altri registri hanno lo stesso contenuto che avevano al termine dell'esecuzione dell'ultima istruzione del programma.

RESET: La funzione del tasto RESET consiste nel riportare il Nanocomputer al suo stato iniziale. Si riprende dall'inizio l'esecuzione del sistema operativo. Tutti i registri sono azzerati e tutti i precedenti indirizzi di breakpoint sono cancellati.

INTERRUTTORE CASS-TTY: Per i Nanocomputer interfacciati sia con un registratore audio che con una teletype, questo interruttore seleziona uno dei due tipi di dispositivi per l'ingresso e l'uscita seriale.

UNITA' CENTRALE DI ELABORAZIONE (CPU)

Il Nanocomputer è un microcomputer basato sul microprocessore Z-80. Lo Z-80, contenuto in un package dual-in-line a 40 pin (DIP), fu progettato e prodotto dalla Zilog Corporation nel 1976. E' prodotto ora anche dalla SGS-ATES in Europa.

CLOCK

Il cristallo di quarzo che si trova nell'angolo in basso a sinistra della scheda, ha una frequenza di 2,4576 MHz ed è utilizzato per controllare un generatore di clock a tale frequenza appunto (pari a circa 2,5 MHz). Questa frequenza di clock di 2,5 MHz pilota il microprocessore Z-80 attraverso tutte le sue fasi di elaborazione. La massima frequenza che può essere applicata ad uno Z-80A è di 4 MHz. Sfortunatamente, a questa frequenza, i dispositivi PROM come la 2708 o l'EPROM 2716 non sono abbastanza veloci. E' per questo motivo che l'SGS-ATES ha scelto la frequenza standard Z80CPU di 2,5 MHz. Notate che, a 2,5 MHz, uno stato o ciclo di clock ha una durata di 400 nanosecondi, o 0,4 microsecondi.

MEMORIA

La memoria del Nanocomputer è composta di memoria lettura/scrittura dinamica e di memoria a sola lettura. La memoria lettura/scrittura presente sul Nanocomputer è di 4K byte (estendibili a 16K byte) ed è posta nei primi 4K indirizzi, 0000H-0FFFH.

La ROM del Nanocomputer è di 2K byte, espandibili a 8K byte, ed è indirizzata negli ultimi 2K o 8K, a seconda delle dimensioni della ROM stessa.

PORTE DI I/O

Il Nanocomputer permette un I/O sia parallelo che seriale.
Due chip Z80-PIO implementano l'I/O parallelo, mentre un dispositivo di interfaccia seriale e la circuiteria di interfacciamento verso la cassetta audio, implementano l'I/O seriale. Un PIO è utilizzato per l'interfacciamento con il display e la tastiera, l'altro è disponibile per l'utilizzatore per sue specifiche applicazioni.
Il dispositivo di interfaccia seriale può essere utilizzato per interfacciare il Nanocomputer con terminali seriali, mentre la circuiteria di interfacciamento verso la cassetta audio serve appunto per il collegamento con un registratore a cassette.

DESCRIZIONE DELLA BASETTA PER ESPERIMENTI (BREADBOARD)

In molti esperimenti che farete con il vostro Nanocomputer, vi sarà richiesto di costruire dei circuiti elettrici usando una basetta per esperimenti (breadboard), del filo, dei circuiti integrati, ed altri componenti. Qui di seguito vi diamo una breve descrizione di una basetta. Più avanti, svolgeremo più a fondo questo argomento.
La basetta è progettata in modo da permettervi di realizzare i molti esperimenti descritti nei capitoli successivi. I circuiti integrati, le resistenze, i condensatori e i fili saranno collegati direttamente alla basetta.
Nelle figure riportate più sotto sono presentate due vedute della basetta per esperimenti, una inferiore e una superiore.
La basetta contiene 128 gruppi di cinque terminali, elettricamente collegati, per connessioni senza saldatura, divisi in due gruppi di 64 e posti ai lati di una banda centrale. Inoltre vi sono 8 gruppi di 25 terminali simili sui bordi della scheda. Il termine "senza saldatura" viene usato poichè è possibile con tali terminali fare connessioni elettriche tra componenti elettronici senza dover effettuare saldature.

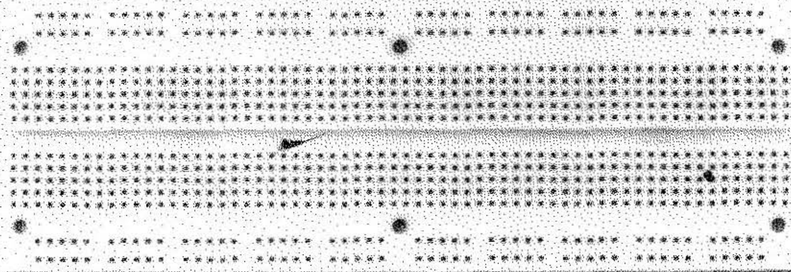


Figura 4-1. Vista superiore della basetta.

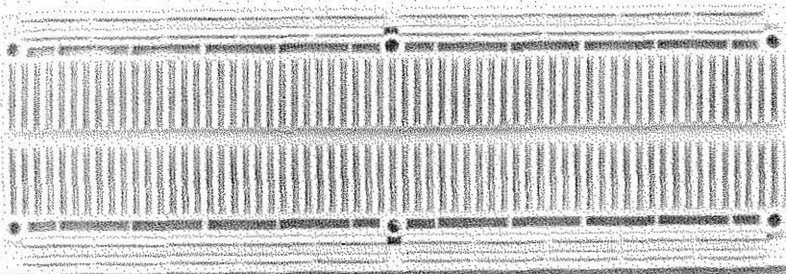


Figura 4-2. Vista inferiore della basetta.

I gruppi centrali di cinque terminali connessi elettricamente possono contenere i circuiti integrati, lasciando ancora liberi, per ogni piedino di chip a 14 e a 16 piedini, quattro terminali per ulteriori collegamenti. I gruppi di 25 terminali connessi elettricamente, ai bordi della piastra, sono collegati o al + 5V o alla massa. Essi alimentano sia i circuiti integrati che i "moduli di funzioni ausiliarie", che descriveremo in un altro capitolo.

REGOLE PER ESEGUIRE GLI ESPERIMENTI

Nei seguenti capitoli userete il Nanocomputer per eseguire degli esperimenti che mettono in pratica i principi della programmazione e dell'interfacciamento, ma prima di iniziare, vogliamo raccomandarvi di osservare le seguenti rgole base:

1. Pianificate in anticipo gli esperimenti in modo da sapere quali risultati dovreste ottenere.

2. Togliete dalla basetta tutti i collegamenti e i componenti rimasti da precedenti esperimenti e quindi non più necessari.

3. IMPORTANTE: Prima di fare un qualsiasi collegamento, togliete il collegamento di alimentazione (+5 V) tra il connettore sui cui avete disponibili i segnali del bus e la basetta per esperimenti. Notate che non vi abbiamo detto di togliere l'alimentazione a tutto il microcomputer, perchè, così facendo, si cancellerebbe tutta la memoria a lettura/scrittura.

4. Dopo aver staccato il collegamento +5 V dalla basetta, collegate con attenzione i circuiti di interfaccia al microcomputer. Prima di fare qualsiasi altro collegamento, collegate l'alimentazione ai singoli circuiti integrati.

5. Fate molta attenzione a dove mettete i diversi chip sulla basetta; una disposizione attenta di questi dispositivi può ridurre di molto la "giungla" dei fili di collegamento, presente anche in circuiti di modesta complessità.

6. Provate i circuiti che avete collegato per essere sicuri che funzionino bene. *Fate molta attenzione nell'eseguire i collegamenti di alimentazione ai circuiti integrati.* Infatti, se sono sbagliati, potreste "bruciare" il vostro chip e forse anche cancellare la memoria a lettura/scrittura. Per vedere se i circuiti si scaldano, metteteci sopra un dito; se sono molto caldi vuol dire che c'è qualcosa di sbagliato.

7. Collegate alla basetta la tensione +5 V quando tutto è già stato controllato; a questo punto potete fare il test del dito per verificare se qualche chip si scalda troppo.

8. Una volta terminato l'esperimento, non scollegate i circuiti, ma prima guardate se l'esperimento successivo non richiede l'impiego degli stessi circuiti già collegati.

9. Staccate l'alimentazione principale del microcomputer solo alla fine della giornata di lavoro. Se avete interfacciato una cassetta al Nanocomputer, potete registrare il vostro programma prima di togliere l'alimentazione.

COME SONO PRESENTATE LE ISTRUZIONI PER GLI ESPERIMENTI

Le istruzioni per eseguire ogni esperimento sono presentate nel modo seguente:

SCOPO

Tutto quello che viene presentato sotto questo titolo serve a definire lo scopo dell'esperimento, sarebbe quindi meglio, ogni volta che fate un esperimento, ricordare per quale motivo è stato proposto.

CONFIGURAZIONI DEI PIN DEI CIRCUITI INTEGRATI

Sotto questo titolo vengono date, con il permesso della SGS-ATES, le configurazioni dei pin per tutti i circuiti integrati usati negli esperimenti. Tutti gli esperimenti usano i chip Low Power Schottky della SGS-ATES. Nel caso in cui venga usato lo stesso circuito integrato dell'esperimento precedente, può non essere riportata la configurazione dei pin.

SCHEMA DEL CIRCUITO

Ogni volta, vi verrà fornito lo schema di tutto il circuito che userete nell'esperimento e voi dovrete analizzarlo per capirne bene il funzionamento prima di iniziare. Controllate i numeri dei pin di tutti i collegamenti ai circuiti integrati e *ricordate che i collegamenti di alimentazione alle porte (GATE) sono omessi.* Fate molta attenzione alle connessioni con l'alimentazione +5 V e massa perché, se ne dimenticate qualcuna, il circuito non vi funzionerà.

PROGRAMMA

Vi sarà fornito il programma del microcomputer che voi dovrete caricare all'indirizzo di memoria che vi sarà indicato.

PASSI

Sotto i titoli PASSO 1, PASSO 2, PASSO 3, etc., sono spiegate in dettaglio le istruzioni su quello che dovete fare in quella particolare fase di esperimento. Durante l'esecuzione dell'esperimento vi saranno poste delle domande cui voi dovrete rispondere subito, apponendo le risposte negli spazi bianchi indicati. Dopo aver scritto la risposta, confrontatela con la risposta che vi viene data. Se la vostra risposta non è esatta, vi sarà certamente possibile capirne il motivo; quindi, prima di continuare, correggetela.

DOMANDE

Spesso vi saranno fatte delle domande per controllare: (a) quanto avete capito dell'esperimento appena concluso (b) la vostra capacità di anticipare i futuri esperimenti o problemi (c) la vostra abilità nel mettere in relazione il testo con le nozioni apprese direttamente dagli esperimenti e nel rispondere, basandovi sulle nozioni acquisite, a domande che si riferiscano ad argomenti non ancora trattati nel testo. Il numero delle domande che vi saranno fatte dipenderà soprattutto dal tipo di esperimento. In alcuni capitoli, le domande saranno raggruppate in un RIEPILOGO. Nel RIEPILOGO vi saranno pure date le risposte alle domande.

UN ATTIMO DI ATTENZIONE

A tutti i programmatori novizi che ci stanno seguendo vogliamo chiarire molto bene quanto segue:

È impossibile danneggiare un microcomputer con un programma sbagliato.

Potete, per sbaglio, cancellare il contenuto della memoria a lettura/scrittura, ma non potete danneggiare o distruggere il microcomputer avendo degli errori nel programma e tentando di farlo eseguire, quindi rilassatevi e divertitevi col vostro microcomputer, fate pure errori di programmazione e imparate da essi.

Potete però danneggiare il microcomputer Z-80 se:

- Applicate l'alimentazione in modo errato.
- Toccate, anche incidentalmente, con degli oggetti metallici i collegamenti sulla piastra di circuito stampato. La scheda del microcomputer è provvista di un rivestimento che riduce in parte il problema.
- Sbagliate a collegare i fili in un interfacciamento. In particolare state molto attenti ad entrare sul bus dati; *per inserire delle linee di dati servitevi di buffer THREE-STATE.*
- Lo fate cadere.
- Lavorate in un ambiente troppo caldo o corrosivo.
- Cercate di effettuare una riparazione senza sapere bene come si fa.

Molti strumenti di laboratorio sono racchiusi in involucri di metallo o di plastica molto dura e sono protetti, fino ad un certo punto, da un uso disattento; invece il Nanocomputer NBZ80 è privo di involucro per potervi permettere di vedere come è costruito e come funziona; quindi è molto vulnerabile. Noi crediamo sia importante che non vi lasciate intimidire dal microcomputer, non mettetelo perciò in un involucro non trasparente.

L'NBZ80-S invece, è posto in un involucro ed è principalmente dedicato ad esperimenti in cui costruite voi stessi i vostri circuiti.

Ricordate quindi: se state solamente eseguendo un programma, non potete, in alcun modo, danneggiare il microcomputer; se invece state sia programmando che interfacciando il microcomputer dovete stare attenti. In alcuni casi, potete danneggiare con un programma un chip Z-80 se non usate correttamente un circuito di interfaccia.

Vi preghiamo perciò di essere attenti.

INTRODUZIONE AGLI ESPERIMENTI

Questi esperimenti si propongono di mostrare il funzionamento dei diversi tasti della tastiera (keyboard). Per condurre questi esperimenti dovete avere:

- Un Nanocomputer NBZ80 o NBZ80-S funzionante.
- Una tastiera per il Nanocomputer ed il relativo software di gestione su PROM/ROM (fornito col Nanocomputer).
- Un alimentatore per l'NBZ80 (l'NBZ80-S ha l'alimentatore incluso).

Gli esperimenti che eseguirete possono essere riassunti nel modo seguente:

Esperimento N.	Commenti
1	Illustra come operano i tasti numerici da 0 a F, il tasto con freccia a sinistra e quello con freccia a destra sulla tastiera del Nanocomputer.
2	Illustra la funzione dei tasti ST e 2ND quando i dati vengono caricati nei registri.
3	Illustra come un'informazione presente nel DATA DISPLAY può essere caricata in una specifica locazione di memoria lettura/scrittura. E' illustrata anche la funzione del tasto INC, tasto particolarmente utile per visualizzare il contenuto di locazioni di memoria poste in successione.
4	Illustra come caricare ed eseguire passo passo (tasto SS) un semplicissimo programma. Illustra anche la funzione dei tasti GO e RESET.
5	Mostra due routine di utilità del Nanocomputer che sono residenti in ROM: test della RAM e test della tastiera e del display.

ESPERIMENTO N. 1

Scopo

Questo esperimento illustra il funzionamento dei tasti numerici, del tasto con freccia a sinistra e di quello con freccia a destra sulla tastiera del Nanocomputer.

Passo 1

Date l'alimentazione al Nanocomputer e premete il tasto RESET. Dovreste osservare quattro digit esadecimali sul display degli indirizzi, due digit esadecimali sul display dei dati, e la luce di selezione dovrebbe essere nella posizione PC. Nel nostro caso, sul display dell'indirizzo abbiamo letto 0000 e sul display dei dati 00.

Passo 2

Premete il tasto 0. Che cosa osservate nel display dei dati?

Noi abbiamo osservato che sul digit di destra è presente uno zero, mentre gli altri digit sono spenti. Se a voi non è successa la stessa cosa, significa che qualcosa non va nel vostro Nanocomputer. Dovreste controllarlo prima di continuare (si veda l'esperimento N. 5 di questo capitolo relativamente ai test di autodiagnosi della memoria e della tastiera/display).

Passo 3

Premete il tasto 1 e poi il tasto 2. Lo 0 dovrebbe essersi spostato verso sinistra di due digit per fare posto all'1 e al 2. Quindi il display dei dati dovrebbe visualizzare uno spazio vuoto seguito da 012. Inserite 3, poi 4. Che cosa osservate adesso?

Noi abbiamo osservato che lo 0 è stato tolto dal display per fare posto al valore 4. Il display dei dati ha quattro digit illuminati: 1234.

Passo 4

Continuate ad inserire valori esadecimali. Ogni volta che viene inserito un valore, il display di dati a quattro digit si sposta verso sinistra mentre scompare il valore che era visualizzato del digit di sinistra. Che cosa è successo al display d'indirizzo? E al selettore luminoso?

Entrambi sono rimasti invariati.

Passo 5

Ora premete una sola volta il tasto con freccia verso destra. Che cosa osservate?

Noi abbiamo osservato che cambiano parecchie cose. Innanzitutto, sia il display di indirizzo che quello di dati hanno cambiato contenuto, e su di essi si legge ora 0000 e 00, rispettivamente. Può darsi che il vostro Nanocomputer non riporti esattamente la stessa lettura, ma comunque dovrebbe essere cambiato qualcosa. Inoltre, il selettore luminoso è ora spostato sulla posizione MEM. Quanto vedete vuol dire che la locazione di memoria 0000 contiene il byte di dati 00.

Passo 6

Premete il tasto con freccia a sinistra. Dovrebbe venire ripristinato di nuovo quanto era presente sul display all'inizio dell'esperimento. Nel nostro caso avevamo 0000 00. Ciò significa che il registro PC a 16 bit contiene 0000 e che il contenuto della locazione di memoria 0000 è 00.

Passo 7

Come fareste a portare la luce di selezione nella posizione AF?
Vi sono due modi per farlo, usando il tasto con freccia a sinistra, oppure usando quello con freccia a destra. In entrambi i casi, non dovete fare altro che premere il tasto e tenerlo premuto finchè raggiungete la posizione desiderata. L'indicazione luminosa avanza passo passo, illuminando ogni led di selezione finchè raggiunge AF.

Passo 8

Che cosa osservate sui display degli indirizzi e dei dati quando il selettore è nella posizione AF?

Noi abbiamo osservato (e lo stesso dovreste vedere voi) un display degli indirizzi abbuiato e un display dei dati con visualizzati i quattro digit 0000. Questo rappresenta il contenuto della coppia di registri AF, cioé l'accumulatore contiene 00 e i flag contengono 00.

Può darsi che il vostro display dei dati contenga quattro cifre esadecimali diverse.

Passo 9

Spostate ancora la luce di selezione. Cosa potete dire dei display degli indirizzi e dei dati per le posizioni IR, AF, BC, DE,HL? E delle posizioni IX, IY, SP, PC, e MEM? E della posizione I/O?

Noi abbiamo osservato che le posizioni IR, AF, BC, DE e HL non danno luogo ad alcuna indicazione sul display degli indirizzi mentre sul display dei dati appaiono quattro digit che rappresentano il contenuto della coppia di registri selezionata. Con le posizioni IX, IY, SP, PC e MEM si dovrebbe vedere un display degli indirizzi a quattro digit e un display dei dati a due digit. Per le posizioni IX, IY, SP e PC il display degli indirizzi dà il contenuto del registro a 16 bit selezionato, mentre il display dei dati dà il contenuto della locazione di memoria indirizzata dal registro. Il significato del display per la posizione MEM è già stato dato nel Passo 5.
Nella posizione I/O, compaiono due cifre nel display degli indirizzi e in quello dei dati. Il display degli indirizzi contiene un codice di dispositivo mentre il display dei dati rappresenta il byte di informazione che è contenuto in quel momento in quel dispositivo. Nei capitoli successivi, parleremo a lungo dell'I/O e dei codici di dispositivo.

ESPERIMENTO N. 2

Scopo

Questo esperimento illustra l'utilizzo dei tasti ST e 2ND per caricare dei dati nei registri.

Passo 1

Spostate la luce di selezione alla posizione BC, inserite i digit esadecimali 11 nel display dei dati, e premete il tasto ST. Che cosa osservate?

Dovreste vedere che nel dispay dei dati, che visualizza il contenuto della coppia di registri BC, compare 11 nei digit più a destra. In altre parole avete memorizzato un byte (esadecimale 11 o binario 00010001) nel registro C della CPU Z-80!

Passo 2

Ora tentate di inserire 22 nel display dei dati, poi premete il tasto 2ND seguito dal tasto ST. Che cosa si legge ora sul display dei dati? Quale registro dello Z-80 è stato cambiato questa volta?

Sul display si dovrebbe leggere 2211. Stavolta avete memorizzato un byte nel registro di ordine alto, cioè il registro B, della coppia di registri BC.

Passo 3

Come memorizzereste i byte 2103 nella coppia di registri HL?

Risposta: Premete i seguenti tasti uno dopo l'altro.

1. Freccia a destra, finchè il selettore luminoso è in posizione HL
2. 2
3. 1
4. 2ND
5. ST, per memorizzare il byte di ordine alto 21 in H
6. 0
7. 3
8. ST, per memorizzare il byte di ordine basso 03 in L

Esiste qualche altro modo per fare la stessa cosa?
La risposta è sì; il tasto usato per posizionare la luce di selezione avrebbe potuto essere quello con freccia a sinistra e il byte di ordine basso (03) avrebbe potuto essere memorizzato in L prima che il byte di ordine alto (21) fosse memorizzato in H. Quindi, grazie alle caratteristiche della tastiera del Nanocomputer, è possibile effettuare in più modi anche operazioni abbastanza semplici, come, ad esempio, caricare dei dati nei registri.

Passo 4

Provate a caricare dei dati in qualche altro registro. In particolare, provate con uno dei registri non appaiati, come IX, IY, SP. Notate che, se inserite due digit e premete ST, il registro a 16 bit viene caricato con 00 a sinistra e con il byte (a due digit) che voi avete inserito sulla destra. Ora tentate di inserire quattro digit esadecimali, per esempio ABCD, e premete ST. Che cosa succede? Riuscite a spiegarlo?

Vediamo che ABCD compare sul display degli indirizzi, il che significa che il registro a 16 bit è stato caricato con due nuovi byte (AB e CD). In effetti quattro digit (due byte) vengono SEMPRE memorizzati con la luce di selezione a IX, IY, SP.
Se inserite solo due digit, il Nanocomputer colloca due zeri sulla sinistra. Che succede se inserite un digit e premete ST? O tre digit e premete ST?

Si è verificato quanto viene chiamato LEFT ZERO FILLING (aggiunta di zeri a sinistra). Ciò evita di introdurre gli zeri di sinistra.

ESPERIMENTO N. 3

Scopo

Questo esperimento illustra come si possono caricare dati nelle locazioni di memoria e come si può usare il tasto INC per visualizzare il contenuto delle locazioni di memoria successive.

Passo 1

Dato che lavoriamo con le locazioni di MEMoria e con il loro contenuto, posizionate il selettore luminoso a MEM. Come abbiamo già detto in precedenza, i quattro digit del display degli indirizzi danno l'indirizzo di memoria e il display dei dati dà il contenuto di quella locazione.
Guardiamo il contenuto della locazione 0100. Per fare questo inserite 0100 o 100 tramite la tastiera, poi premete LA, il tasto di Load Address.
Che cosa osservate?

Dovreste notare che 0100 compare ora sul display degli indirizzi. Nel nostro caso, sul display dei dati si legge 00. Sarebbe a dire che nel nostro caso la locazione 0100 conteneva 8 bit tutti a 0, cioé il byte 00. Il vostro dato potrebbe essere differente.

Passo 2

Tentate di memorizzare AA nella locazione 0100. Inserite AA, poi premete ST. Che cosa è successo?

Sul display degli indirizzi si dovrebbe leggere 0101 e, nel nostro caso, abbiamo osservato che sul display dei dati si leggeva 00.

Che cosa significa questo? Significa che AA è già stato memorizzato alla locazione 0100 e il computer aspetta che voi memorizziate un byte alla locazione seguente 0101. Il contenuto attuale di tale locazione (0101) compare sul display dei dati (sul nostro si leggeva 00). Tentate di memorizzare BB inserendo BB e poi premendo ST.

Passo 3

Non fidiamoci troppo del nostro Nanocomputer. Dovremmo fare un controllo per assicurarci che 0100 contenga AA e che 0101 contenga BB. Come possiamo farlo? Per vedere il contenuto 0100, è necessario visualizzare 0100 come indirizzo. Per questo bisogna usare il tasto LA. Impostate 0100 o 100 seguito da LA. Speriamo che vediate AA sul display di dati.

Passo 4

E per vedere il contenuto di 0101? Potreste inserire l'indirizzo 0101 (o 101) e premere quindi il tasto LA. Il Nanocomputer può però anche INCrementare di uno l'indirizzo di una locazione di memoria, e si può così evitare di impostare nuovamente gli indirizzi. Premete una volta leggermente il tasto INC. Che cosa vedete?

Il registro degli indirizzi è aumentato di 1 e il registro dei dati visualizza BB. Premete INC e tenetelo premuto. Come potete vedere, le locazioni di memoria vengono mostrate in sequenza.
I tasti LA, ST e INC sono quelli che userete di più in questo libro, sia per caricare dei programmi per lo Z-80 sia per verificare il caricamento stesso. Potrete vedere nel prossimo esperimento quanto questi tasti siano importanti.

Passo 5

Riportate la luce di selezione ad una delle posizioni IR, AF, BC, DE, HL, IX, IY, SP o PC, inserite 11 e premete LA. Che cosa osservate?
Si è acceso il led verde che indica la presenza di un errore! Vi abbiamo portato fuori strada. Usando il tasto LA con la luce di selezione che non indica MEM o I/O, avete tentato di eseguire un'operazione illegale. L'unico caso in cui si può usare LA è per caricare nel display degli indirizzi un indirizzo di MEMoria o un codice di dispositivo di I/O. Il led di errore si spegne automaticamente quando, iniziando una nuova operazione da tastiera, premerete un tasto.

Passo 6

Provate a indovinare come si fa a caricare un codice di dispositivo ad un byte nel display degli indirizzi, (i contenuti delle porte di I/O compaiono automaticamente sul display di dati).

La procedura è la seguente: posizionate la luce di selezione su I/O, inserite due digit esadecimali, e premete LA.

ESPERIMENTO N. 4

Scopo

Questo esperimento carica ed esegue, sia alla massima velocità che passo-passo, un programma molto semplice. Viene illustrato l'uso dei tasti SS, GO, RESET.

Passo 1

Alla fine di questo passo, vogliamo che le locazioni di memoria del Nanocomputer che vanno da 0100 a 0105 contengano i seguenti valori:

Indirizzi	Contenuti
0100	3E
0101	00
0102	3C
0103	C3
0104	02
0105	01

Per arrivare a questo, posizionate la luce di selezione su MEM. Inserite 0100 o 100 e premete LA. Sul display degli indirizzi si dovrebbe leggere 0100, che significa che il Nanocomputer è pronto per iniziare a caricare un programma da 0100. Il programma verrà caricato eseguendo queste operazioni:

3E	premete ST
00	premete ST
3C	premete ST
C3	premete ST
02	premete ST
01	premete ST

Passo 2

Dovete ora verificare se avete caricato il programma in modo corretto. In che modo potete farlo?

Usate il tasto INC: prima visualizzate la locazione di memoria 0100 (usate il tasto LA), poi premete il tasto INC per visualizzare tutte le successive locazioni di memoria. Assicuratevi che il contenuto di tutte le locazioni di memoria sia esatto.

Passo 3

Caricate il registro PC con l'indirizzo di partenza del programma, cioè 0100.
Per farlo, ricordate di portare prima la luce di selezione su PC, inserite 0100 o 100, e poi premete ST.

Passo 4

Posizionate la luce di selezione su AF. Ciò significa, per il Nanocomputer, che vorreste osservare il contenuto della coppia di registri AF man mano che il programma viene eseguito a bassa velocità. Premete ripetutamente il tasto SS. Che cosa osservate?

Ogni volta che ripremete il tasto SS, dovreste notare che il contenuto del registro A (l'accumulatore) aumenta di uno. Tenete premuto il tasto SS. Il registro A continuerà a conteggiare. Il vostro Nanocomputer stà eseguendo un programma a bassa velocità!

Passo 5

Ora premete GO. Che cosa succede?

Tutti i led e i display si sono spenti. Avete rotto il Nanocomputer? NO! Esso sta ora eseguendo il programma alla massima velocità. Potete fermarlo premendo il tasto RESET.

Quale era la velocità massima di esecuzione del programma? L'accumulatore veniva incrementato più di centomila volte al secondo.
Notate che, subito dopo aver premuto RESET, i display del Nanocomputer, e la luce di selezione si portano in modo da visualizzare PC. In qualunque momento il Nanocomputer si "spenga", come avete avuto modo di osservare, potete sempre riprendere il controllo premendo RESET o BREAK. Avete appena eseguito il vostro primo programma sullo Z-80. Proseguiremo ora mostrandovi in che modo dovrete scrivere i vostri programmi.

ESPERIMENTO N. 5

Scopo

Questo esperimento vi presenta due routine che l'SGS-ATES fornisce insieme al sistema operativo del Nanocomputer nella memoria a sola lettura. Le due routine verificano la memoria a lettura/scrittura e l'hardware e il software della tastiera e del display.

Passo 1

Dato che il programma che eseguirete in questo esperimento risiede già nella memoria a sola lettura, non c'è nessun bisogno di caricare manualmente nessun byte, come avete fatto prima.
Esaminiamo prima la routine di controllo della memoria a lettura/scrittura.
Questo test è diviso in due parti. La prima parte controlla le locazioni di memoria a lettura/scrittura da 0FAB a 0FFF. Queste locazioni di memoria vengono usate dal sistema operativo del Nanocomputer per memorizzare dati, come i byte che devono essere visualizzati sul display.
Ogni locazione di memoria viene controllata scrivendovi degli zeri all'interno, e poi leggendola per vedere se gli zeri sono effettivamente presenti.

Successivamente per ogni locazione di memoria vengono caricati alternativamente, e quindi letti, i seguenti byte:

00000001
00000010
00000100
00001000
00010000
00100000
01000000
10000000

Questo tipo di test "rotazione di un bit attraverso la memoria" è un test comunemente usato. Non si tratta certamente di un test esauriente ma assicura con un'ottima approssimazione la funzionalità della memoria. Il test delle locazioni da 0FAB a 0FFF si verifica automaticamente quando viene premuto il tasto RESET. Se il test non rivela nessun errore, il Nanocomputer mostra di essere pronto accendendo il led del PC e visualizzando il contenuto del registro PC sulla destra.

Premete molte volte il tasto RESET. Speriamo che non venga rivelato nessun errore!

Passo 2 (opzionale)

Vi raccomandiamo di **non** eseguire questo passo se non siete provvisti di dispositivi di memoria dinamica 4027P - 4F1 di riserva, poichè tali dispositivi sono molto sensibili all'elettricità statica e possono essere facilmente danneggiati se tolti dallo zoccolo o inseriti in modo non corretto. Prima di eseguire il passo 2 è importante leggere l'Appendice III sulle precauzioni da osservare quando si maneggiano dispositivi MOS.

Innanzitutto, staccate l'alimentazione al Nanocomputer: potreste infatti danneggiare il computer se la lasciate inserita. Una volta tolta l'alimentazione, e facendo riferimento all'illustrazione di pag. 4-5 togliete con attenzione il chip n. 1 della memoria RAM dal suo zoccolo di plastica. Ci riuscirete molto più facilmente aiutandovi con un piccolo cacciavite. STATE ATTENTI A NON PIEGARE I PIN DEL CHIP DI MEMORIA. Una volta rimosso questo chip, ridate la corrente al Nanocomputer e premete il tasto RESET. Che cosa osservate?
Osserviamo che si accende il led ERR e sul display si legge

— — — — — — — 8

Questa uscita insolita è prodotta dalla routine di controllo della memoria che ha rivelato un errore nel chip n. 1. La presenza dell'8 (tutti i segmenti accesi) corrisponde al chip sbagliato. Analogamente, togliendo il chip 3 della RAM si produrrebbe un 8 nella terza posizione (ammesso che gli altri chip fossero a posto). Dato che è facilissimo rompere i pin sui chip, non vi consigliamo di verificare con un esperimento la corrispondenza fra la posizione del chip e la posizione dell'8 sul display da cui risulta l'errore.

TOGLIETE LA CORRENTE AL NANOCOMPUTER e rimettete a posto, con attenzione, il chip della memoria. Una volta che il chip della RAM è di nuovo a posto sul suo socket, ridate la corrente al Nanocomputer e premete il tasto RESET.
Non dovreste rilevare nessun errore.

Passo 3

La seconda parte del test della memoria a lettura/scrittura controlla la RAM dell'utente, locazioni da 0000 a 0FAA.

Questo test di memoria può essere eseguito premendo questi tasti:

1. Assicuratevi che la luce di selezione indichi PC (usate i tasti con freccia a destra e a sinistra)
2. Caricate l'indirizzo MEMTUT (Appendice IV)
3. Premete il tasto GO

Avete così lanciato il programma che inizia nella locazione di memoria MEMTUT, che, ovviamente è il programma di controllo della memoria. Mentre il computer fa ruotare i bit attraverso la memoria, il display dovrebbe restare spento. Questo potrebbe continuare per sempre. Il display si accenderà solo se si verifica uno di questi avvenimenti:

1. Premete il tasto RESET o BREAK, che mette fine al test e provoca un ritorno al normale stato operativo con l'indicatore luminoso in posizione PC.

2. Il test della memoria rivela una locazione di memoria sbagliata. In questo caso, sul display vedrete quanto segue:

L'indirizzo del byte sbagliato nei quattro digit a sinistra
Il byte di dati scritto nei due digit successivi
Il byte di dati letto nei due digit successivi
Naturalmente, il test ha rilevato errore perchè le ultime due coppie di digit non erano uguali.

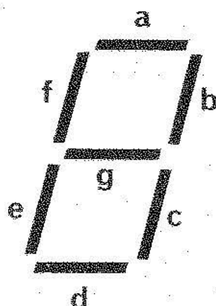
Passo 4

Il test della tastiera/display, controlla tutti i tasti tranne due (BREAK e RESET) e tutti i LED presenti sulla tastiera/display del Nanocomputer. Il programma inizia alla locazione di label CONTST (vedi App. IV). Quindi,per eseguirlo,

1. Posizionate la luce di selezione su PC
2. Premete i tasti corrispondenti all'indirizzo della label CONTST
3. Premete il tasto GO

Che cosa osservate?

Si accendono per prima cosa tutti i segmenti del display e tutte le luci di selezione. Premendo uno qialunque dei tasti (ad eccezione di BREAK e RESET) i display e i LED di selezione si spengono; vengono poi accese, in posizione legata alla fila orizzontale cui il tasto premuto appartiene, due luci di selezione e un segmento di ogni display. Premendo un tasto appartenente ad un'altra fila si accenderanno altre luci di selezione e un altro gruppo di segmenti.



Vi mostriamo ora la corrispondenza esistente fra la fila dei tasti, a partire dall'alto, e i led accesi ed i segmenti:

Fila dei tasti	LED Acceso	Segmento Acceso
1	BRK, IR	a
2	IY, ARS	g
3	ERR, IX	f
4	SP, HL	e
5	PC, DE	d
6	BC, MEM	c
7	AF, I/O	b

I tasti BREAK e RESET non possono essere controllati perchè, premendo uno o l'altro, ponete termine al test. Verificate la corrispondenza suddetta premendo parecchi tasti di ogni fila orizzontale sulla tastiera del Nanocomputer.

CAPITOLO 5

ALCUNI SEMPLICI PROGRAMMI DEL MICROCOMPUTER Z-80

INTRODUZIONE

In questo capitolo caricherete ed eseguirete parecchi semplici programmi che impiegano le istruzioni dello Z-80 trattate nel Cap. 3.

OBIETTIVI

Alla fine di questo capitolo sarete in grado di:

- Dare la definizione dei seguenti termini: codice binario, codice esadecimale, codice assembler, linguaggio ad alto livello.
- Spiegare l'operatività delle seguenti istruzioni dello Z-80: NOP; INC A; HALT; LD A, <B2>; LD (<B3><B2>),A; JP <B3><B2>
- Caricare ed eseguire dei semplici programmi del microprocessore Z-80 sul Nanocomputer
- Leggere e capire i listing di programma dello Z-80 in linguaggio assemblatore, i quali mostrano la locazione di memoria, il codice oggetto, il codice sorgente e i commenti.

RIEPILOGO DI ALCUNE ISTRUZIONI DELLO Z-80

Nel Capitolo 3, abbiamo parlato delle seguenti istruzioni dello Z-80:

Codice macchina esadecimale	Codice mnemonico	Operazione
00	NOP	;Nessuna operazione
3C	INC A	;Incrementa di 1 l'accumulatore
76	HALT	;Alt del microcomputer
3E	LD A,< B2 >	;Carica il byte di dato immediatamente seguente
		;nell'accumulatore
<B2>		
32	LD (<B3><B2>),A	;Memorizza il contenuto dell'accumulatore nella
<B2>		;locazione di memoria indirizzata dai due byte
<B3>		;seguenti di questa istruzione a tre byte
C3	JP <B3><B2>	;Salta in modo incondizionato all'indirizzo di
<B2>		;memoria indicato dai due byte seguenti di questa
<B3>		;istruzione a tre byte

LINGUAGGI DI PROGRAMMAZIONE E LISTING DEI PROGRAMMI

Avete letto nel Capitolo 2 che un programma è una serie di istruzioni e che le istruzioni si presentano in diverse forme: binarie, esadecimali, codici mnemonici e parole complete (linguaggio ad alto livello). Prendiamo in considerazione queste forme.

Codice binario: Questo è il vero linguaggio dello Z-80. Alla fine, tutte le istruzioni dei programmi devono essere espresse in questa forma per risultare comprensibili allo Z-80. Se l'umanità considerasse questo un modo naturale di esprimersi, non ci sarebbe alcun bisogno di altre forme per presentare le istruzioni al computer. Ogni forma, esadecimale, mnemonica, linguaggio ad alto livello, è tagliata sempre più su misura per l'uomo e sempre meno per il computer. Ovviamente, questa convenienza sempre maggiore per l'uomo ha un suo prezzo, e cioè il tempo che è necessario per eseguire la traduzione in binario, e lo spazio di memoria richiesto per ospitare un programma che possa eseguire la traduzione.

Prendiamo un programma semplice, scritto in codice binario. Questo programma addiziona i due numeri contenuti nelle locazioni di memoria 0160 H e 0161 H e memorizza la somma nella locazione 0162 H:

Un listing di programma in codice binario

```
0 0 1 1 1 0 1 0
0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 1 1 1
0 0 1 1 1 0 1 0
0 1 1 0 0 0 0 1
0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0
0 0 1 1 0 0 1 0
0 1 1 0 0 0 1 0
0 0 0 0 0 0 0 1
```

Questa rappresentazione può piacere al vostro Nanocomputer, ma è sicuramente molto poco naturale per voi.

Codice esadecimale: Questo modo di rappresentare le istruzioni è migliore rispetto al metodo binario perchè abbrevia ogni gruppo di 8 bit, in due digit esadecimali. Rappresentiamo il programma già indicato usando il codice esadecimale:

Un listing di programma in codice esadecimale

```
3A
60
01
47
3A
61
01
80
32
62
01
```

Questo è senz'altro un miglioramento rispetto alle nostre esigenze umane, ma uno Z-80 non sarebbe in grado di interpretare questo programma così com'è. E' quindi necessario un programma *loader* (caricatore) *esadecimale* che converta il suddetto listing esadecimale nel codice binario comprensibile per lo Z-80. Il sistema operativo del

Nanocomputer contiene un loader esadecimale che "legge" quando viene premuto un tasto esadecimale (0-F) e converte il codice esadecimale in codice binario per memorizzarlo nella memoria di lettura/scrittura.
Quali sono i vantaggi e svantaggi del loader esadecimale? Ecco i vantaggi:

1. Maggiore facilità di interfacciamento programmatore/macchina, per il fatto che si ha a che fare con due sole cifre invece che con otto cifre per ogni byte

2. Maggiore efficienza da parte del programmatore, per il fatto che gli errori vengono più facilmente scoperti e corretti.

Gli svantaggi sono invece:

1. Il loader esadecimale, essendo di per sè un programma, deve risiedere in memoria in modo che il dato esadecimale possa essere interpretato (cioè convertito in rappresentazione binaria) e memorizzato.

2. La conversione porta via del tempo.

Ai giorni nostri, i programmatori stanno diventando più costosi dei computer, per cui, in molti casi, il fattore umano viene considerato più importante e, molto probabilmente, lo diventerà sempre di più.

Codice mnemonico: Questa forma di rappresentare le istruzioni ci porta di un ulteriore passo più lontani dal computer e più vicini al programmatore. Il codice mnemonico usa i caratteri alfabetici per descrivere le istruzioni. Per esempio LD B,A è il codice mnemonico per l'istruzione di codice esadecimale 47. Questa istruzione fa sì che il contenuto del registro A venga caricato nel registro B. Chiaramente, la rappresentazione mnemonica è più leggibile di quella esadecimale (per lo meno da un punto di vista umano). Ecco il listing (cioé l'elenco delle istruzioni) mnemonico per il solito programma addizione:

Un listing mnemonico di programma o listing di assemblaggio

```
LD     A,(0160H)
LD     B,A
LD     A,(0161H)
ADD    A,B
LD     (0162H),A
```

Notate che: a. I codici mnemonici si usano per descrivere le operazioni che lo Z-80 deve eseguire. Ad esempio, il codice mnemonico "LD" è un'abbreviazione di "LOAD" che è un'istruzione che sposta i dati da una sorgente ad una destinazione nella forma generale:

LD "destinazione", "sorgente"

La prima istruzione carica il registro A o accumulatore con il contenuto della locazione di memoria 0160H.

b. Sono stati assegnati dei nomi ai registri interni alla CPU dello Z-80. Per esempio, sono menzionati nel programma suddetto i registri A e B (noti di solito come 111 e 000 allo Z-80).

Un programma scritto in codice mnemonico viene chiamato programma in linguaggio assembler. I codici mnemonici del linguaggio assembler usati nel programma suddetto, vennero definiti dalla Zilog Corporation quando venne sviluppata la CPU dello Z-80 e sono questi i codici mnemonici che l'SGS-ATES raccomanda di usare. Comunque, indubbiamente la SGS-ATES non può costringere l'utente ad usare questi codici, e molte altre Società hanno sviluppato altri tipi di codici mnemonici alternativi, che secondo loro erano migliori. Per lo Z-80, il codice mnemonico universale più diffuso è quello della SGS-ATES e della Zilog, quindi sarà quello che useremo in questo libro.

Come potete ben immaginare, la procedura di traduzione di un programma dal linguaggio assembler al codice binario è molto complicata. Questo è vero, ma la procedura è così sistematica e ripetitiva che può essere programmata e implementata sullo Z-80 stesso. Il programma che accetta in ingresso un listing in assembler (detto codice sorgente) e pone in uscita un programma in codice binario (detto codice oggetto) si chiama assemblatore (o assembler). L'SGS-ATES ha scritto un programma assemblatore, per una versione migliorata del Nanocomputer, che è attualmente disponibile memorizzato su audiocassette (denominato ASS-Z), oppure memorizzato su ROM (denominato FR-Z). Per usare l'assemblatore è necessaria una versione più potente del Nanocomputer, una tastiera con caratteri ASCII (l'alfabeto, la virgola, i numeri, ecc.) per inserire il codice mnemonico ed un display ASCII per metterlo in uscita. Supporremo che voi non abbiate un microcomputer così completo.

Perciò in questo libro l'assemblaggio dei programmi verrà eseguito manualmente.

Assemblaggio manuale significa che il programmatore prende il set di codici mnemonici e li traduce uno per uno in codice esadecimale.

Il codice esadecimale puo essere inserito nel Nanocomputer in cui il loader esadecimale esegue la conversione finale in codice binario intelleggibile allo Z-80. La procedura di assemblaggio manuale fa largo uso di tabelle di conversione mnemonico-esadecimale, in cui tutte le istruzioni (codici mnemonici) dello Z-80 sono ordinate per gruppi a seconda del tipo di funzione che realizzano. Tali tabelle sono naturalmente disponibili e, essendo un ottimo aiuto per il programmatore, sono inserite in questo libro.

Vi mostriamo di seguito un listing del solito programma assemblato manualmente. Notate che il codice esadecimale qui viene chiamato codice oggetto, e il codice assembler si chiama codice sorgente.

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
01 50	3A 60 01	LD A,(0160H)	;A = contenuto della ;locazione di memoria ;0160
01 53	47	LD B,A	;Carica A in B
01 54	3A 61 01	LD A,(0161H)	;A = contenuto della ;locazione 0161
01 57	80	ADD A,B	;Somma A e B
01 58	32 62 01	LD (0162H),A	;Memorizza la somma in ;0162

Potete quindi vedere che l'assemblaggio manuale implica la traduzione dei codici mnemonici cui va aggiunto l'indirizzo della locazione, nella memoria a lettura/scrittura, del dato su cui l'istruzione opera. Nel nostro caso il programma inizia alla locazione 0150 e finisce a 015A. I commenti sono ovviamente facoltativi, ma largamente consigliabili. Questo è il modo secondo cui verranno presentati tutti i programmi di questo libro.

Linguaggi ad alto livello: L'ultima categoria di linguaggi di programmazione di cui parleremo è quella dei linguaggi ad alto livello. Tali linguaggi si allontanano di un passo ulteriore dal computer rispetto al linguaggio assembler. In genere, essi non richiedono dei programmatori che sappiano qualcosa circa i registri o gli indirizzi di memoria. Piuttosto, questi linguaggi sono ideati per permettere ad un programmatore di concentrarsi sul problema da risolvere invece che sul computer.

Per esempio, in un linguaggio ad alto livello come il FORTRAN, il programma per sommare due numeri apparirebbe, molto semplicemente, in questo modo:

```
RISULT = X + Y
```

Esempi di altri linguaggi ad alto livello sono COBOL, PL/1, ALGOL, SNOBOL, PASCAL, JOVIAL, e molti altri. Ognuno di questi linguaggi richiede dei grossi programmi traduttori chiamati compilatori, che convertano i programmi in codice oggetto binario.

Su una versione più potente del Nanocomputer si può utilizzare un linguaggio ad alto

livello chiamato BASIC. Il BASIC SGS-ATES è un linguaggio orientato alle applicazioni di controllo ed è disponibile in una versione PROM/ROM di 8K (BAS-Z). Man mano che viene inserita una linea di programma in linguaggio BASIC, il trasduttore BASIC interpreta la linea, la converte in (molti) byte di codice oggetto binario e poi la esegue immediatamente. Non è come un compilatore FORTRAN o COBOL o come l'assemblatore dello Z-80, che aspetta finchè non è stato inserito tutto il programma prima di iniziare la traduzione. Per questo motivo il traduttore BASIC è chiamato *interprete.* La versione BASIC del nostro programma di addizione è

LET A1 = X + Y.

I linguaggi ad alto livello hanno due grossi vantaggi, che spesso compensano e superano lo svantaggio di dover utilizzare grossi compilatori o interpreti.

1. I programmi sono orientati più verso la procedura da effettuare che verso il computer.
2. I programmi scritti in un linguaggio ad alto livello per un computer, spesso possono venire eseguiti su di un altro computer con varianti minime o nulle, proprietà questa che viene chiamata PORTABILITY (l'essere trasferibile). Cio non è quasi mai possibile con programmi scritti in codice assembler o esadecimale, per il fatto che sono in correlazione troppo stretta con la macchina.

PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLER

Come abbiamo detto in precedenza, in questa serie di capitoli programmeremo in linguaggio assembler e assembleremo manualmente i programmi per ottenere il codice esadecimale da caricare nel Nanocomputer.
Oltre alla traduzione dei codici mnemonici in esadecimali, l'assemblaggio manuale implica il dover collocare il programma in qualche punto della memoria a lettura/scrittura del vostro Nanocomputer. Guardate la prima colonna del listing che vi abbiamo mostrato nell'ultimo paragrafo. Questa colonna, intitolata "Locazione di memoria", indica l'indirizzo di memoria del primo byte di ogni linea di programma. Se la linea è occupata da più di un byte, gli indirizzi sucessivi a quello indicato contengono tali byte, e la linea di programma seguente inizia alla locazione di memoria successiva.

Proviamo ora ad eseguire alcuni semplici programmi Z-80 sul Nanocomputer.

INTRODUZIONE AGLI ESPERIMENTI

I seguenti esperimenti vi permettono di eseguire parecchi semplici programmi, di cui, in ogni esperimento, vi è data una descrizione dettagliata.
Acquisterete così esperienza nel caricare ed eseguire i programmi del microprocessore, ed imparerete anche alcune nozioni fondamentali sulla programmazione. Gli esperimenti che eseguirete si possono così riassumere:

Esperimento N.	Commenti
1	Illustra l'esecuzione delle istruzioni NOP, HALT e LD A,<B2>.
2	Illustra l'esecuzione delle istruzioni INC A e JP <B3> <B2> in un semplice loop di programma.
3	Illustra l'esecuzione di un programma con un semplice loop, e delle istruzioni INC B e INC C.

4 Illustra l'istruzione LD (<B2><B3>),A che serve a trasferire in memoria un certo valore.

5 Illustra l'esecuzione del programma di addizione usato come esempio nel paragrafo che parlava dei linguaggi di programmazione e dei listing.

ESPERIMENTO N. 1

Scopo

Questo esperimento illustra l'esecuzione di tre istruzioni dello Z-80: NOP, HALT e LD A, <B2>.

Programma N. 1

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
0100	3E BB	LD A,BBH	;Carica l'accumulatore ;con BB
0102	76	HALT	;Alt

Programma N. 2

0103	3E BB	LD A,BBH	;Carica l'accumulatore ;con BB
0105	00	NOP	;Nessuna operazione
0106	3E FF	LD A,FFH	;Carica l'accumulatore ;con FF
0108	76	HALT	;Alt

Passo 1

Date alimentazione al Nanocomputer e premete il tasto RESET per attivare la CPU dello Z-80. Posizionate la luce di selezione su MEM, inserite 0100 e premete LA.
Siete ora pronti per iniziar a caricare un programma in memoria, partendo dalla locazione 0100. Inserite il programma N. 1, alternativamente inserendo i byte del codice oggetto e premendo il tasto ST.

Passo 2

Ricontrollate, per essere sicuri che il programma N. 1 sia stato caricato correttamente, inserendo l'indirizzo di memoria 0100 (inserite 0100 seguito da LA) e premendo il tasto INC.

Passo 3

Prima di passare all'esecuzione di questo programma, fermiamoci a riflettere su cio che ci aspettiamo che succeda. Scrivete nello spazio che segue quale dovrebbe essere il contenuto dell'accumulatore dopo che l'esecuzione del programma è terminata.

Fatta la vostra previsione, procediamo ora per scoprire se è esatta.
Per prima cosa controlliamo l'accumulatore per scoprire cosa contiene ora: posizionate la luce di selezione su AF. Quali sono le due cifre a sinistra che si leggono sul display dei dati?

Passo 4

Eseguite il programma partendo da 0100: posizionate la luce di selezione su PC, inserite 0100, ST, poi premete GO! Che cosa succede?

Il display del Nanocomputer si spegne immediatamente! Il computer è fermo.

Passo 5

Riattivate il Nanocomputer premendo il tasto BREAK. Il computer "si risveglia" immediatamente con la luce di selezione sulla posizione PC, che ci fa vedere il contenuto (0103) del contatore di programma cioé del registro PC. Questa locazione contiene l'istruzione successiva che dovrebbe essere eseguita. Poiché la locazione 0102 contiene un'istruzione di HALT (76), l'istruzione successiva non è stata raggiunta.

Passo 6

Controlliamo il contenuto dell'accumulatore, per vedere se la previsione che avete fatto al Passo 3 era esatta: spostate la luce di selezione su AF.
Qual è il contenuto di A?

Speriamo di vedere BB. La prima istruzione del programma N. 1 fa sì che il microcomputer inserisca il valore BB nel registro A, l'accumulatore. L'istruzione che segue, che arresta il microprocessore non ha alcun effetto sul registro A, quindi, dopo l'esecuzione del programma, il contenuto di A dovrebbe essere BB.

Passo 7

Cambiamo il contenuto di A in 00 (00,2ND, ST) ed eseguiamo quindi il Programma N. 1 passo-passo, guardando l'accumulatore: posizionate la luce di selezione su PC, inserite 0100, premete ST, posizionate la luce di selezione su AF, e premete una volta SS. Che cosa succede?

Il contenuto del registro A varia immediatamente da 00 a BB.

Passo 8

Premete ancora SS. Noterete che non succede niente. Il meccanismo del passo-passo non permette al computer di fermarsi. Perciò, in modo passo-passo, il comando di HALT non ha alcun effetto.

Passo 9

Caricate e controllate il Programma N. 2 partendo dalla locazione 0103. Quale dovrebbe essere il contenuto del registro A una volta ultimata l'esecuzione del programma?

Passo 10

Eseguite il Programma N. 2. Assicuratevi ancora di aver riattivato il Nanocomputer usando il tasto BREAK (non usate RESET, dato che modifica il contenuto di tutti i registri). Controllate il registro A.

Speriamo che il contenuto che leggete sia FF. Quindi quello che è successo è che il valore originale BB è stato sostituito da FF.

Passo 11

Eseguiamo questo programma in modo passo-passo per osservare l'effetto dell'istruzione NOP, alla locazione 0105. Caricate il PC con 0103 e quindi posizionate la luce di selezione su AF per osservare l'esecuzione passo-passo. Premete SS: il registro A viene caricato immediatamente con BB. Premete SS: non succede niente. Questa è l'esecuzione dell'istruzione NOP. Se posizionate la luce di selezione su PC, vedrete che sul display degli indirizzi si legge 0106, che sta ad indicare che 0105, cioè NOP, è appena stata eseguita, e che l'istruzione seguente è quella che inizia a 0106. Selezionate ancora AF e premete di nuovo SS.
In A viene trasferito immediatamente FF, mostrando l'effetto dell'istruzione LD A,FFH.

Avete quindi verificato che NOP ha un solo effetto su di un programma, e cioè fa sì che il programma non faccia niente per un passo.

ESPERIMENTO N. 2

Scopo

Lo scopo di questo esperimento è di illustrare l'esecuzione delle istruzioni INC A e JP <B3><B2> in un semplice loop di programma.

Programma N. 3

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
0109	3E FF	LD A,FFH	;Carica l'accumulatore ;con FF
010B	3C	INC A	;Incrementa ;l'accumulatore
010C	C3 0B 01	JP 010BH	;Salta all'indirizzo 010B

Passo 1

Notate che l'istruzione INC A è un'istruzione ad un byte, che dice al computer di sommare uno al contenuto dell'accumulatore. L'istruzione JP fa sì che il controllo venga trasferito in modo incondizionato all'indirizzo dato dai due byte che seguono il codice operativo C3.
Notate che i due byte d'indirizzo compaiono con il byte d'indirizzo LO per primo, e con il byte d'indirizzo HI per secondo.

Passo 2

Caricate e verificate il Programma N. 3.

Passo 3

Prendiamo in esame il Programma N. 3 per anticipare quello che farà. Il programma N. 3 carica l'accumulatore con FF e quindi incrementa A di 1. La terza istruzione provoca un salto incondizionato all'indietro alla locazione 010B, cioè all'istruzione INC A. Vi è così un incremento dell'accumulatore seguito da un salto indietro all'istruzione INC A. L'effetto di tutto questo è che l'istruzione INC A viene eseguita ripetutamente finchè qualcuno non interrompe il loop spegnendo il computer, o premendo RESET o BREAK.

Passo 4

Ora eseguiamo il programma passo passo per vedere che cosa succede all'accumulatore (A) e al contatore di programma (PC). Osserviamo prima l'accumulatore, quindi, dopo aver caricato il PC con l'indirizzo d'inizio del programma, 0109, spostate la luce di selezione sulla posizione AF. Ricordate che i due digit a sinistra che compaiono sul display dei dati rappresentano il contenuto del registro A. Premete una volta il tasto SS. Che cosa succede al registro A?

Abbiamo osservato che il suo contenuto è diventato immediatamente FF.

Passo 5

L'istruzione successiva dice di incrementare A. Quale dovrebbe essere il contenuto di A dopo il passo successivo?

Premete una volta SS e verificate la vostra idea. Dovreste osservare che il contenuto di A è diventato 00. Avete osservato anche un cambiamento del registro F? Di questo parleremo più avanti.

Passo 6

Tenete premuto per un po' il tasto SS. Dovreste osservare un incremento del contenuto del registro A ogni volta che l'istruzione INC A viene eseguita.

Passo 7

Ora posizionate la luce di selezione su PC. Tenete premuto il tasto SS ed osservate che cosa succede al contenuto del registro PC. Scrivete quello che avete osservato:

Abbiamo osservato che il registro PC passava alternativamente fra 010B e 010C.

ESPERIMENTO N. 3

Scopo

Viene illustrata l'esecuzione di un programma con un semplice loop che contiene le istruzioni INC B e INC C.

Programma N. 4

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
010F	04	INC B;	;Incrementa il registro B
0110	C3 20 01	JP 0120H;	;Salta alla locazione 0120
0120	C3 30 01	JP 0130H;	;Salta alla locazione 0130
0130	04	INC B;	;Incrementa il registro B
0131	0C	INC C	;Incrementa il registro C
0132	C3 0F 01	JP 010FH;	;Salta alla locazione 010F

Passo 1

Facciamo alcune osservazioni sul listing del Programma N. 4.

a. Per prima cosa notate che un'istruzione di salto come JP 0130H si traduce in una serie di tre byte esadecimali:

   C3 Codice operativo dell'istruzione di salto
   30 Byte d'indirizzo LO
   01 Byte d'indirizzo HI

   C'è sempre la tentazione di tradurre l'istruzione con i byte d'indirizzo nell'ordine opposto. Sembra che tutti i programmatori cadano in questa trappola almeno una volta. Tutto quello che possiamo fare è continuare a mettervi in guardia.

b. In questo programma compaiono due istruzioni nuove, INC B e INC C. Esse fanno sì che il computer addizioni 1 ai registri B e C, rispettivamente.

Passo 2

Caricate e verificate il Programma N. 4. Notate che è sufficiente che carichiate le locazioni specificate perchè non vengono mai usate altre locazioni di memoria fra 010F e 0134.
Il caricamento di questo programma richiede un'ottima conoscenza da parte vostra dei tasti LA e ST.

Passo 3

Quando siete convinti di avere caricato correttamente il Programma N. 4, studiate molto attentamente le linee del programma in codice mnemonico e cercate di stabilire esattamente che cosa sta facendo il programma.

Descriviamo quello che fa questo programma, prima a parole, e poi con una figura.

1. Il primo passo di programma incrementa il registro B, cioè somma 1 al suo contenuto attuale.

2. Il passo successivo è un salto incondizionato alla locazione 0120. Questo significa che questo passo porta il computer alla locazione 0120 per l'istruzione successiva.

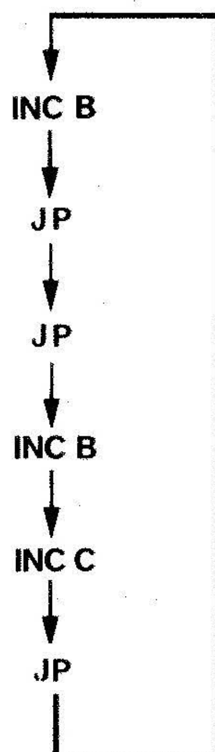
3. Ora il computer arriva alla locazione 0120 e legge l'istruzione. Un altro salto incondizionato! Dove andiamo questa volta? A 0130.

4. Ed eccoci a 0130, che succede ora? Il programma dice di incrementare B, per cui 1 viene sommato al registro B.

5. L'istruzione seguente dice di incrementare il registro C, perciò, viene sommato 1 a C.

6. L'istruzione successiva è un salto, quindi ci trasferiamo alla locazione 010F. Notate che 010F è l'istruzione eseguita al punto 1. Perciò abbiamo un ciclo (loop): vengono eseguiti in continuazione i passi da 1 a 6 finché il programma non viene interrotto dall'esterno.

Disegnamo una figura:



Il risultato finale di tutti questi incrementi e salti è che ad ogni ciclo il registro B viene incrementato due volte, mentre il registro C una volta sola.

Passo 4

Eseguiamo ora il Programma N. 4 passo-passo. Vogliamo osservare che cosa succede al registro PC e alla coppia di registri BC durante l'esecuzione. Prima caricate la coppia di registri BC con degli zeri per inizializzarli entrambi (posizionate la luce di selezione su BC, inserite 00, 2ND, ST per inizializzare B, e inserite 00, ST per inizializzare C). Ora caricate il registro PC con 010F, e lasciando la luce di selezione su PC, premete SS. Quante volte premerete SS prima che PC legga 0120, 0130, e poi ancora 010F?

Le vostre risposte dovrebbero essere 2, 3 e 6 rispettivamente. Abbiamo quindi un loop di programma a 6 passi. Quando terminerà questo loop? Mai, a meno che non lo fermiamo premendo RESET o BREAK.

Passo 5

Tenete premuto il tasto SS e guardate la configurazione ricorrente dei sei indirizzi di PC. Inserite questi indirizzi nello spazio che segue:

Noi abbiamo osservato per PC i seguenti valori:

010F
0110
0120
0130
0131
0132

Passo 6

Osserviamo ora la coppia di registri BC durante l'esecuzione passo passo del Programma N. 4. Avevamo previsto nel Passo 3 che il registro B sarebbe stato incrementato due volte per ogni volta che veniva incrementato il registro C. Vediamo se questo è vero. Leggete la coppia BC e scrivete quello che vedete (assicuratevi di essere all'inizio di un nuovo ciclo, cioè PC = 010F).

Abbiamo osservato 04 02. (Il valore che leggerete voi potrà essere diverso e dipenderà da quanto tempo avrete tenuto premuto il tasto SS nel Passo 5). Ovvero: B è stato incrementato da 00 a 04, mentre C è stato incrementato da 00 a 02. Questo confermerebbe le nostre congetture. Per esserne sicuri, incominciate a premere SS, e scrivete le vostre osservazioni:

Abbiamo osservato quanto segue:

Dopo la linea di programma 1: B è stato incrementato
Dopo la linea di programma 4: B è stato incrementato di nuovo
Dopo la linea di programma 5: C è stato incrementato

Le linee di programma 2, 3 e 6 sono occupate da istruzioni di salto.

Passo 7

Tenete premuto per un po' il tasto SS. Noi abbiamo osservato la seguente sequenza sul display di dati:

08 04
09 04
0A 04
0A 05
0B 05
0C 05
0C 06

0D 06
0E 06
0E 07
0F 07
10 07
10 08

Passo 8

Facciamo un'ultima osservazione su questo programma. Le prime due istruzioni di salto possono essere sostituiteda una sola istruzione di salto. Quale?

La risposta è JP 0130H. Il motivo per cui ciò avviene è che il primo JP provoca sempre un salto all'istruzione a 0120, che è ancora un salto a 0130.

ESPERIMENTO N. 4

Scopo

Questo esperimento illustra l'istruzione LD (<B3> <B2>),A che serve a scrivere un byte in una locazione di memoria.

Programma N. 5

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
0136	3E 11	LD A,11H;	;Carica A con 11 (hex)
0138	32 45 01	LD (0145H), A;	;Carica la locazione di ;memoria 0145 con il ;contenuto di A
013B	76	HALT;	;Alt

Passo 1

L'istruzione nuova è l'istruzione LD (0145H),A posta all'indirizzo 0138. Questa istruzione copia il contenuto del registro A nella locazione di memoria 0145. Notate che, come l'istruzione di salto, l'indirizzo che costituisce il secondo e il terzo byte di questa istruzione LD ha il byte LO che precede il byte HI.

Passo 2

Caricate e verificate il Programma N. 5.

Passo 3

Controllate l'attuale contenuto della locazione di memoria 0145 e mettetelo per iscritto. Noi abbiamo osservato 00.

Passo 4

Eseguite il programma suddetto o alla massima velocità (GO) o passo-passo (SS). Se lo eseguirete alla massima velocità, dovrete ridare il controllo al sistema operativo del Nanocomputer premendo BREAK per far sì che i registri e la memoria vengano conservati. Controllate l'accumulatore e la locazione di memoria 0145. Che cosa contengono?

Dovrebbero contenere entrambi 11.

Passo 5

Scrivete, caricate ed eseguite un programma, partendo dalla locazione 013C, che serva a memorizzare 22H nella locazione di memoria 0146. Potete controllare il funzionamento del programma guardando alla locazione 0146 dopo che avete eseguito il programma stesso. Qui sotto, abbiamo scritto la nostra soluzione. Ovviamente la vostra può essere un pò diversa.

Soluzione

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
013C	3E 22	LD A,22H	;Carica 22H in A
013E	32 46 01	LD (0146H),A	;Carica con A la locazione ;di memoria 0146 H
0141	76	HALT	;Alt

ESPERIMENTO N. 5

Scopo

Questo esperimento illustra l'esecuzione del programma di somma usato come esempio nel paragrafo di questo capitolo che tratta i linguaggi di programmazione e i listing dei programmi.

Programma N. 6

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
0150	3A 60 01	LD A,(0160H)	;A = contenuto della ;locazione 0160
0153	47	LD B,A	;Carica il registro B con A
0154	3A 61 01	LD A,(0161H)	;A = contenuto della ;locazione 0161
0157	80	ADD B	;Somma B ad A, ;memorizza il risultato ;in A

0158	32 62 01	LD (0162H),A	;Memorizza la somma ;nella locazione 0162
015B	76	HALT	;Alt

Passo 1

Notate che in questo programma sono state introdotte molte istruzioni nuove. Tali istruzioni vengono qui di seguito elencate e spiegate. Nei capitoli seguenti, verranno trattate tutte nei minimi particolari.

Codice oggetto	Codice mnemonico	Operazione
3A <B2> <B3>	LD A,(<B3> <B2>)	;Carica il contenuto della locazione di ;memoria <B3> <B2> nell'accumulatore
47	LD B,A	;Carica il contenuto dell'accumulatore nel ;registro B
80	ADD B	;Somma il contenuto del registro B al contenuto ;del registro A memorizzando la somma ;risultante nel registro A

Passo 2

Caricate e verificate il Programma N. 6.

Passo 3

Proviamo ora il programma sommando 2 e 3. Per farlo, dobbiamo memorizzare 2 nella locazione 0160 e 3 nella locazione 0161 (potremmo memorizzare 2 nella locazione 0161 e 3 nella locazione 0160, non fa nessuna differenza). Di conseguenza memorizzeremo 02H e 03H (gli equivalenti esadecimali di 2 e 3, rispettivamente). Ora caricate il registro PC con 0150 e premete GO.

Passo 4

Premete BREAK per riportare il controllo al sistema operativo del Nanocomputer e guardate quindi il contenuto del registro A e della locazione di memoria 0162. Che cosa vedete?

Ci aspettiamo di vedere 05, ed in effetti è quello che troviamo.

Passo 5

Eseguite il programma passo passo guardando il contenuto dei registri A e B.

Passo 6

Cercate di sommare altre coppie di numeri. Per non avere noie, assicuratevi che la

somma dei due numeri dati non sia maggiore di FFH, ovvero di 255 (decim.) maggiore cioè della capacità di un byte ad otto bit.

Passo 7

Se ve la sentite, provate a intuire che cosa succede quando la somma supera 255.

DOMANDE RIEPILOGATIVE

Le domande seguenti vi saranno di aiuto per ripassare le istruzioni che avete eseguito in questo capitolo.

1. Spiegate che cosa fa ognuna delle operazioni seguenti:

   a. 3E
      5B

   b. C3
      A5
      03

   c. 3C

   d. 32
      E4
      1B

   e. 76

   f. 00

2. Facendo riferimento al testo di questo capitolo o all'Appendice B, date l'esatto codice esadecimale per le operazioni seguenti:

   a. Saltare all'indirizzo di memoria HI = 24 e LO = 53
   b. Memorizzare il contenuto dell'accumulatore nella locazione di memoria HI = 02 e LO = 38
   c. Caricare in modo immediato il byte di dati 92 nell'accumulatore
   d. Incrementare il contenuto dell'accumulatore
   e. Arrestare il microcomputer
   f. Nessuna operazione

RISPOSTE

1. a. Sposta il byte di dati 5B nell'accumulatore
   b. Salta all'indirizzo di memoria HI = 03 e LO = A5
   c. Incrementa il contenuto dell'accumulatore
   d. Memorizza il contenuto dell'accumulatore nella locazione di memoria HI = 1B e LO = E4
   e. Alt del microcomputer
   f. Nessuna operazione

2. a. C3
   53
   24

   b. 32
   38
   02

   c. 3E
   92

   d. 3C

   e. 76

   f. 00

CAPITOLO 6

REGISTRI, MEMORIA E TRASFERIMENTO DATI

INTRODUZIONE

In questo capitolo, imparerete alcuni fra i molti modi di trasferire i dati fra il microprocessore Z-80 e la memoria, nonchè tra locazioni diverse in memoria. Potrete dare una prima occhiata all'intero set di istruzioni dello Z-80, che vi colpirà per lo meno per la sua mole e la sua complessità. Vengono presentate le istruzioni JP NZ, INC e DEC, in modo che possiate creare LOOP per generare ritardi di tempo (DELAY).

OBIETTIVI

Alla fine di questo capitolo, sarete in grado di:

- Capire cosa vuol dire *decodificare un'istruzione*
- Indicare il codice binario a tre bit che definisce ogni registro non specializzato
- Dare la definizione del termine *modo d'indirizzamento*
- Dare la definizione di: indirizzamento tra registri
  indirizzamento immediato
  indirizzamento immediato esteso
  indirizzamento indiretto tramite registri
  indirizzamento esteso
- Illustrare le istruzioni di incremento e decremento dei registri
- Illustrare le istruzioni LD per i diversi modi d'indirizzamento
- Illustrare le istruzioni di trasferimento blocchi
- Scrivere un programma che contenga un loop di delay
- Scrivere programmi per diversi tipi di trasferimento dei dati

SET DI ISTRUZIONI DELLO Z-80

Nelle pagine seguenti vi presentiamo il set di (cioè l'insieme delle) istruzioni dello Z-80, in una forma simile a quella suggerita da R. Baker per il microcomputer 8080A della Intel. Questo tipo di presentazione del set di istruzioni è apparsa per la prima volta su Byte, una rivista dedicata a coloro che hanno l'hobby del microcomputer. Nel Capitolo 3, avete imparato che il codice operativo (op. code) è il codice che indica l'operazione specifica che il microprocessore esegue. Con otto bit di informazione (un codice operativo di un byte), è possibile avere 2 all'ottava potenza, o 256 diversi codici operativi: questi codici sono quelli che vedete nella Tabella 6-1: "Codici operativi ad un Byte dello Z-80". Le cinque cifre binarie della colonna a sinistra sono i primi cinque bit del codice operativo a otto bit. I tre bit rimanenti sono posti in testa ad ogni colonna. Le istruzioni con un codice operativo a due e a tre byte sono mostrate nelle tabelle successive.
Non vi verrà richiesto di imparare a memoria questo set di istruzioni. Il nostro scopo è semplicemente quello di mostrarvi l'intero set, in modo che possiate farvi riferimento man mano che imparate delle istruzioni nuove. Per esempio, nel Capitolo 3, avete imparato le seguenti istruzioni:

Codice oggetto	Codice sorgente
00	NOP
32 <B2> <B3>	LD (<B3> <B2>),A
3C	INC A
3E <B2>	LD A,<B2>
76	HALT
C3 <B2> <B3>	JP <B3> <B2>

Riuscite a trovarli nelle tabelle? (N.B.: queste sono istruzioni con un codice operativo ad un byte).
Le tabelle delle istruzioni, (Tabelle 6-1, 6-2, 6-3, 6-4 e 6-5), offrono una panoramica delle possibilità dello Z-80.
In particolare, si può facilmente notare come il set di istruzioni dell'Intel 8080A costituisca una base per il linguaggio più esteso dello Z-80. I codici operativi ad un byte sono implementati tutti, tranne dodici, sull'8080A. Le nuove istruzioni in codice operativo ad un byte dello Z-80 sono l'istruzione dello scambio dei registri (EXX) e le istruzioni di "salto relativo" (JR) che sono indicate con un asterisco (*) nella Tabella 6-1.
Gli altri codici operativi, che non vengono usati dall'8080A, sono CB, DD, ED e FD. Ognuno di essi è sempre il primo byte di un'istruzione con un codice operativo a più byte dello Z-80. Più avanti parleremo di tutti questi codici istruzioni, ma prima parleremo soprattutto delle istruzioni con un codice operativo ad un byte.

Se esaminate i codici operativi ad un byte della Tabella 6-1, noterete che i primi due bit determinano la classe generale dell'operazione. Tutte le istruzioni di caricamento (LD) ad un solo byte della Tabella 6-1 hanno 01 come primi due bit. Tutte le istruzioni aritmetiche e logiche iniziano con le cifre binarie 10. Ad eccezione delle istruzioni di salto relativo (JR) (che iniziano tutte con 00), le istruzioni di salto, chiamata di subroutine e rientro hanno come primi bit 11. Determinando la relazione esistente fra i singoli bit del codice operativo e le operazioni che vengono eseguite, voi decodificate il codice operativo. In sostanza, questo è quello che il decodificatore di istruzioni fa, elettronicamente, all'interno del microprocessore.

La decodifica dei primi due bit del codice operativo delle istruzioni dello Z-80 (istruzioni in codice operativo ad un byte), è solo un'inizio di analisi; si può andare più oltre. Se esaminate attentamente le istruzioni di caricamento, in cui i primi due bit sono 01, noterete che:

1. Ogni istruzione trasferisce il contenuto di un registro in un altro registro

2. Ogni registro ha un codice di tre bit ad esso associato

Registro	Codice binario
B	000
C	001
D	010
E	011
H	100
L	101
(HL)	110 (vedi nota)
A	111 (vedi nota)

Nota: ricordate che (HL) si riferisce alla locazione di memoria indirizzata dal contenuto della coppia di registri HL. La lettera A si riferisce al registro accumulatore, di cui abbiamo già parlato. Per essere precisi, (HL) non è un registro, ma spesso vi si fa riferimento come se lo fosse. Faremo attenzione a specificare se vogliamo o meno includere (HL) quando useremo la parola "registro".

Per vedere in che modo il codice a tre bit dei registri si applica alle istruzioni di caricamento, consideriamo questo byte:

0 1 1 1 1 0 1 0

Possiamo vedere subito che cosa fa questa istruzione decodificando i suoi bit in questo modo:

```
01- - - - - -  istruzione LD
   111 - - -  A è il registro destinazione
         010  D è il registro sorgente
```

Quindi il codice mnemonico di questa istruzione è LD A,D (controllate sulla Tabella 6-1), il che significa che lo Z-80 trasferisce il contenuto del registro D nell'accumulatore, cioè il registro A è caricato con il contenuto del registro D.
Anche le istruzioni aritmetiche e logiche (quei codici operativi che iniziano con 10) decodificano i registri. Per es. nelle istruzioni del gruppo

da 1 0 0 0 0 - - -
a 1 0 1 1 1 - - -

gli ultimi tre bit corrispondono al registro coinvolto. Così per le istruzioni nei gruppi

da 0 0 0 0 0 1 0 0
a 0 0 1 1 1 1 0 0

e da 0 0 0 0 0 1 0 1
a 0 0 1 1 1 1 0 1

e da 0 0 0 0 0 1 1 1
a 0 0 1 1 1 1 1 1

il terzo, quarto e quinto bit (numerandoli da sinistra a destra) rappresentano il registro che è coinvolto nell'operazione.
Fin qui, abbiamo parlato solo delle istruzioni con un codice operativo ad un byte. Nelle Tabelle 6-2, 6-3, 6-4 e 6-5 potete vedere le istruzioni con codici operativi multi-byte. Nessuno di questi codici viene implementato sul microprocessore Intel 8080A. Tutto il set dell'8080A contiene solo codici operativi ad un byte e, ad eccezione delle istruzioni già citate, appare nella Tabella 6-1. Percio, la maggior parte delle istruzioni nuove dello Z-80 è presentata nelle tabelle dei codici operativi multi-byte (Tab. 6-2÷6-5).

I codici operativi multi-byte si possono dividere in quattro gruppi, a seconda del loro primo byte - CB, DD, ED o FD. Illustreremo ora alcune delle caratteristiche di ognuno di questi gruppi.

Le istruzioni CB (Tabella 6-2)

La Tabella 6-2 mostra i codici operativi a due byte in cui il primo byte è CB. Gli otto bit derivati dalla colonna orizzontale e verticale dell'istruzione rappresentano il secondo byte del codice operativo. Per esempio, l'istruzione BIT 2,C corrisponde al codice operativo a due byte:

Byte 1 = CB
Byte 2 = 51 (0 1 0 1 0 0 0 1)

La composizione del byte 2 è facilmente deducibile dalla Tabella 6-2.

Primi due bit =

00 - - - - - - un'istruzione di rotate o di shift in cui gli ultimi tre bit sono il codice per il registro coinvolto

01 - - - - - - un'istruzione BIT in cui gli ultimi tre bit specificano il registro e gli altri tre bit specificano il bit che dev'essere indirizzato

10 - - - - - - un'istruzione RES in cui gli ultimi sei bit hanno lo stesso significato di quelli dell'istruzione BIT

11 - - - - - - un'istruzione SET in cui gli ultimi sei bit hanno lo stesso significato di quello dell'istruzione BIT

Le istruzioni DD e FD (Tabelle 6-3 e 6-4)

Le istruzioni che iniziano con DD o FD possono avere codici operativi a due o a tre byte. La Tabella 6-3 mostra le istruzioni DD con il codice operativo a due byte. Notate che coinvolgono tutte il registro indice IX. I codici a due byte FD, analogamente alle precedenti, coinvolgono il registro indice IY.
Tutti i codici operativi a tre byte implementati sullo Z-80 hanno la seguente struttura:

Byte 1: DD o FD a seconda del registro indice (IX o IY)
Byte 2: CB
Byte 3: byte di spiazzamento (displacement)
Byte 4: I primi due bit indicano rotate/shift, BIT, RES o SET come nelle suddette istruzioni CB. Gli ultimi tre bit sono sempre 110. I bit rimanenti indicano il tipo di rotazione o di spostamento o il numero del bit, come nelle già indicate istruzioni CB.

La Tabella 6-4 mostra i codici operativi DD a tre byte, dove il byte quattro è ricostruibile dalla colonna orizzontale e verticale dell'istruzione.

Le istruzioni ED (Tabella 6-5)

Le istruzioni che iniziano con ED hanno tutte codici operativi a due byte. La Tabella 6-5 mostra queste istruzioni, dove il secondo byte è derivabile dalla posizione nella colonna orizzontale e verticale dell'istruzione. A chi fosse interessato lasciamo per esercizio il compito di spiegare le diverse configurazioni.

	000	001	010	011	100	101	110	111
00 000	NOP	LD BC,<B3><B2>	LD (BC),A	INC BC	INC B	DEC B	LD B,<B2>	RLCA
00 001	* EX AF,AF'	ADD HL,BC	LD A,(BC)	DEC BC	INC C	DEC C	LD C,<B2>	RRCA
00 010	* DJNZ <B2>	LD DE,<B3><B2>	LD (DE),A	INC DE	INC D	DEC D	LD D,<B2>	RLA
00 011	* JR <B2>	ADD HL,DE	LD A,(DE)	DEC DE	INC E	DEC E	LD E,<B2>	RRA
00 100	* JR NZ,<B2>	LD HL,<B3><B2>	LD (<B3><B2>),HL	INC HL	INC H	DEC H	LD H,<B2>	DAA
00 101	* JR Z,<B2>	ADD HL,HL	LD HL,(<B3><B2>)	DEC HL	INC L	DEC L	LD L,<B2>	CPL
00 110	* JR NC,<B2>	LD SP,<B3><B2>	LD (<B3><B2>),A	INC SP	INC (HL)	DEC (HL)	LD (HL),<B2>	SCF
00 111	* JR C,<B2>	ADD HL,SP	LD A,(<B3><B2>)	DEC SP	INC A	DEC A	LD A,<B2>	CCF

	000	001	010	011	100	101	110	111
01 000	LD B,B	LD B,C	LD B,D	LD B,E	LD B,H	LD B,L	LD B,(HL)	LD B,A
01 001	LD C,B	LD C,C	LD C,D	LD C,E	LD C,H	LD C,L	LD C,(HL)	LD C,A
01 010	LD D,B	LD D,C	LD D,D	LD D,E	LD D,H	LD D,L	LD D,(HL)	LD D,A
01 011	LD E,B	LD E,C	LD E,D	LD E,E	LD E,H	LD E,L	LD E,(HL)	LD E,A
01 100	LD H,B	LD H,C	LD H,D	LD H,E	LD H,H	LD H,L	LD H,(HL)	LD H,A
01 101	LD L,B	LD L,C	LD L,D	LD L,E	LD L,H	LD L,L	LD L,(HL)	LD L,A
01 110	LD (HL),B	LD (HL),C	LD (HL),D	LD (HL),E	LD (HL),H	LD (HL),L	HALT	LD (HL),A
01 111	LD A,B	LD A,C	LD A,D	LD A,E	LD A,H	LD A,L	LD A,(HL)	LD A,A

Tabella 6-1. Codici operativi di Z80 a 1 byte (segue)

	000	001	010	011	100	101	110	111
10 000	ADD A,B	ADD A,C	ADD A,D	ADD A,E	ADD A,H	ADD A,L	ADD A,(HL)	ADD A,A
10 001	ADC A,B	ADC A,C	ADC A,D	ADC A,E	ADC A,H	ADC A,L	ADC A,(HL)	ADC A,A
10 010	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB (HL)	SUB A
10 011	SBC A,B	SBC A,C	SBC A,D	SBC A,E	SBC A,H	SBC A,L	SBC A,(HL)	SBC A,A
10 100	AND B	AND C	AND D	AND E	AND H	AND L	AND (HL)	AND A
10 101	XOR B	XOR C	XOR D	XOR E	XOR H	XOR L	XOR (HL)	XOR A
10 110	OR B	OR C	OR D	OR E	OR H	OR L	OR (HL)	OR A
10 111	CP B	CP C	CP D	CP E	CP H	CP L	CP (HL)	CP A

	000	001	010	011	100	101	110	111
11 000	RET NZ	POP BC	JP NZ,<B3><B2>	JP <B3><B2>	CALL NZ,<B3><B2>	PUSH BC	ADD A,<B2>	RST 0H
11 001	RET Z	RET	JP Z,<B3><B2>	nota 1	CALL Z,<B3><B2>	CALL <B3><B2>	ADC A,<B2>	RST 8H
11 010	RET NC	POP DE	JP NC,<B3><B2>	OUT (<B2>),A	CALL NC,<B3><B2>	PUSH DE	SUB <B2>	RST 10H
11 011	RET C	* EXX	JP C,<B3><B2>	IN A,(<B2>)	CALL C,<B3><B2>	nota 1	SBC A,<B2>	RST 18H
11 100	RET PO	POP HL	JP PO,<B3><B2>	EX (SP),HL	CALL PO,<B3><B2>	PUSH HL	AND <B2>	RST 20H
11 101	RET PE	JP (HL)	JP PE,<B3><B2>	EX DE,HL	CALL PE,<B3><B2>	nota 1	XOR <B2>	RST 28H
11 110	RET P	POP AF	JP P,<B3><B2>	DI	CALL P,<B3><B2>	PUSH AF	OR <B2>	RST 30H
11 111	RET M	LD SP,HL	JP M,<B3><B2>	EI	CALL M,<B3><B2>	nota 1	CP <B2>	RST 38H

Tabella 6-1. Codici operativi di Z80 a 1 byte

Nota 1: Questo è il primo byte di una istruzione di due byte di codice operativo.
N.B.: Le istruzioni contrassegnate da (*) non esistono nel set 8080 A

	000	001	010	011	100	101	110	111
00 000	RLC B	RLC C	RLC D	RLC E	RLC H	RLC L	RLC (HL)	RLC A
00 001	RRC B	RRC C	RRC D	RRC E	RRC H	RRC L	RRC (HL)	RRC A
00 010	RL B	RL C	RL D	RL E	RL H	RL L	RL (HL)	RL A
00 011	RR B	RR C	RR D	RR E	RR H	RR L	RR (HL)	RR A
00 100	SLA B	SLA C	SLA D	SLA E	SLA H	SLA L	SLA (HL)	SLA A
00 101	SRA B	SRA C	SRA D	SRA E	SRA H	SRA L	SRA (HL)	SRA A
00 110	—	—	—	—	—	—	—	—
00 111	SRL B	SRL C	SRL D	SRL E	SRL H	SRL L	SRL (HL)	SLA A

	000	001	010	011	100	101	110	111
01 000	BIT 0,B	BIT 0,C	BIT 0,D	BIT 0,E	BIT 0,H	BIT 0,L	BIT 0,(HL)	BIT 0,A
01 001	BIT 1,B	BIT 1,C	BIT 1,D	BIT 1,E	BIT 1,H	BIT 1,L	BIT 1,(HL)	BIT 1,A
01 010	BIT 2,B	BIT 2,C	BIT 2,D	BIT 2,E	BIT 2,H	BIT 2,L	BIT 2,(HL)	BIT 2,A
01 011	BIT 3,B	BIT 3,C	BIT 3,D	BIT 3,E	BIT 3,H	BIT 3,L	BIT 3,(HL)	BIT 3,A
01 100	BIT 4,B	BIT 4,C	BIT 4,D	BIT 4,E	BIT 4,H	BIT 4,L	BIT 4,(HL)	BIT 4,A
01 101	BIT 5,B	BIT 5,C	BIT 5,D	BIT 5,E	BIT 5,H	BIT 5,L	BIT 5,(HL)	BIT 5,A
01 110	BIT 6,B	BIT 6,C	BIT 6,D	BIT 6,E	BIT 6,H	BIT 6,L	BIT 6,(HL)	BIT 6,A
01 111	BIT 7,B	BIT 7,C	BIT 7,D	BIT 7,E	BIT 7,H	BIT 7,L	BIT 7,(HL)	BIT 7,A

Tabella 6-2. Codici operativi di Z80 a due byte con byte 1=CB (segue)

	000	001	010	011	100	101	110	111
10 000	RES 0,B	RES 0,C	RES 0,D	RES 0,E	RES 0,H	RES 0,L	RES 0,(HL)	RES 0,A
10 001	RES 1,B	RES 1,C	RES 1,D	RES 1,E	RES 1,H	RES 1,L	RES 1,(HL)	RES 1,A
10 010	RES 2,B	RES 2,C	RES 2,D	RES 2,E	RES 2,H	RES 2,L	RES 2,(HL)	RES 2,A
10 011	RES 3,B	RES 3,C	RES 3,D	RES 3,E	RES 3,H	RES 3,L	RES 3,(HL)	RES 3,A
10 100	RES 4,B	RES 4,C	RES 4,D	RES 4,E	RES 4,H	RES 4,L	RES 4,(HL)	RES 4,A
10 101	RES 5,B	RES 5,C	RES 5,D	RES 5,E	RES 5,H	RES 5,L	RES 5,(HL)	RES 5,A
10 110	RES 6,B	RES 6,C	RES 6,D	RES 6,E	RES 6,H	RES 6,L	RES 6,(HL)	RES 6,A
10 111	RES 7,B	RES 7,C	RES 7,D	RES 7,E	RES 7,H	RES 7,L	RES 7,(HL)	RES 7,A

	000	001	010	011	100	101	110	111
11 000	SET 0,B	SET 0,C	SET 0,D	SET 0,E	SET 0,H	SET 0,L	SET 0,(HL)	SET 0,A
11 001	SET 1,B	SET 1,C	SET 1,D	SET 1,E	SET 1,H	SET 1,L	SET 1,(HL)	SET 1,A
11 010	SET 2,B	SET 2,C	SET 2,D	SET 2,E	SET 2,H	SET 2,L	SET 2,(HL)	SET 2,A
11 011	SET 3,B	SET 3,C	SET 3,D	SET 3,E	SET 3,H	SET 3,L	SET 3,(HL)	SET 3,A
11 100	SET 4,B	SET 4,C	SET 4,D	SET 4,E	SET 4,H	SET 4,L	SET 4,(HL)	SET 4,A
11 101	SET 5,B	SET 5,C	SET 5,D	SET 5,E	SET 5,H	SET 5,L	SET 5,(HL)	SET 5,A
11 110	SET 6,B	SET 6,C	SET 6,D	SET 6,E	SET 6,H	SET 6,L	SET 6,(HL)	SET 6,A
11 111	SET 7,B	SET 7,C	SET 7,D	SET 7,E	SET 7,H	SET 7,L	SET 7,(HL)	SET 7,A

Tabella 6-2. Codici operativi di Z80 a due byte con byte 1 = CB
Dalla tabella è ricavabile il valore del secondo byte.

	000	001	010	011	100	101	110	111
00 000								
00 001		ADD IX,BC						
00 010								
00 011		ADD IX,DE						
00 100		LD IX,<B4><B3>	LD (<B4><B3>),IX	INC IX				
00 101		ADD IX,IX	LD IX,(<B4><B3>)	DEC IX				
00 110					INC (IX+<B3>)	DEC (IX+<B3>)	LD (IX+<B3>),<B4>	
00 111		ADD IX,SP						

	000	001	010	011	100	101	110	111
01 000							LD B,(IX+>B3>)	
01 001							LD C,(IX+<B3>)	
01 010							LD D,(IX+<B3>)	
01 011							LD E,(IX+<B3>)	
01 100							LD H,(IX+<B3>)	
01 101							LD L,(IX+<B3>)	
01 110	LD (IX+<B3>),B	LD (IX+<B3>),C	LD (IX+<B3>),D	LD (IX+<B3>),E	LD (IX+<B3>),H	LD (IX+<B3>),L		LD (IX+<B3>),A
01 111							LD A,(IX+<B3>)	

Tabella 6-3. Codici operativi di Z80 a 2 byte con byte 1 = DD (segue)

	Nessuna istruzione nel campo da 000 a 101	110	111
10 000		ADD A,(IX+<B3>)	
10 001		ADC A,(IX+<B3>)	
10 010		SUB (IX+<B3>)	
10 011		SBC A,(IX+<B3>)	
10 100		AND (IX+<B3>)	
10 101		XOR (IX+<B3>)	
10 110		OR (IX+<B3>)	
10 111		CP (IX+<B3>)	

	000	001	010	011	100	101	110	111
11 000								
11 001								
11 010								
11 011								
11 100		POP IX		EX (SP),IX		PUSH IX		
11 101		JP (IX)						
11 110								
11 111		LD SP,IX						

Tabella 6-3. Codici operativi di Z80 a 2 byte con byte 1 = DD

	Nessuna istruzione nel campo da 000 a 101	110	111
00 000		RLC (IX+<B3>)	
00 001		RRC (IX+<B3>)	
00 010		RL (IX+<B3>)	
00 011		RR (IX+<B3>)	
00 100		SLA (IX+<B3>)	
00 101		SRA (IX+<B3>)	
00 110		SRL (IX+<B3>)	
00 111		—	

	Nessuna istruzione nel campo da 000 a 101	110	111
01 000		BIT 0,(IX+<B3>)	
01 001		BIT 1,(IX+<B3>)	
01 010		BIT 2,(IX+<B3>)	
01 011		BIT 3,(IX+<B3>)	
01 100		BIT 4,(IX+<B3>)	
01 101		BIT 5,(IX+<B3>)	
01 110		BIT 6,(IX+<B3>)	
01 111		BIT 7,(IX+<B3>)	

Tabella 6-4. Codici operativi di Z80 a tre byte con byte 1 = DD e byte 2 = CB (segue)

	Nessuna istruzione nel campo da 000 a 101	110	111
10 000		RES 0,(IX+<B3>)	
10 001		RES 1,(IX+<B3>)	
10 010		RES 2,(IX+<B3>)	
10 011		RES 3,(IX+<B3>)	
10 100		RES 4,(IX+<B3>)	
10 101		RES 5,(IX+<B3>)	
10 110		RES 6,(IX+<B3>)	
10 111		RES 7,(IX+<B3>)	

	Nessuna istruzione nel campo da 000 a 101	110	111
11 000		SET 0,(IX+<B3>)	
11 001		SET 1,(IX+<B3>)	
11 010		SET 2,(IX+<B3>)	
11 011		SET 3,(IX+<B3>)	
11 100		SET 4,(IX+<B3>)	
11 101		SET 5,(IX+<B3>)	
11 110		SET 6,(IX+<B3>)	
11 111		SET 7,(IX+<B3>)	

Tabella 6-4. Codici operativi di Z80 a tre byte con byte 1 = DD e byte 2 = CB

	000	001	010	011	100	101	110	111
01 000	IN B,(C)	OUT (C),B	SBC HL,BC	LD (<B4><B3>),BC	NEG	RETN	IM0	LD I,A
01 001	IN C,(C)	OUT (C),C	ADC HL,BC	LD BC,(<B4><B3>)		RETI		LD R,A
01 010	IN D,(C)	OUT (C),D	SBC HL,DE	LD (<B4><B3>),DE			IM1	LD A,I
01 011	IN E,(C)	OUT (C),E	ADC HL,DE	LD DE,(<B4><B3>)			IM2	LD A,R
01 100	IN H,(C)	OUT (C),H	SBC HL,HL					RRD
01 101	IN L,(C)	OUT (C),L	ADC HL,HL					RLD
01 110			SBC HL,SP	LD (<B4><B3>),SP				
01 111	IN A,(C)	OUT (C),A	ADC HL,SP	LD SP,(<B4><B3>)				

	000	001	010	011	100	101	110	111
10 000								
10 001								
10 010								
10 011								
10 100	LDI	CPI	INI	OUTI				
10 101	LDD	CPD	IND	OUTD				
10 110	LDIR	CPIR	INIR	OTIR				
10 111	LDDR	CPDR	INDR	OTDR				

Tabella 6-5. ***Codici*** *operativi di Z80 a due byte con byte 1 = ED*

METODI DI INDIRIZZAMENTO DELLO Z-80

Quasi tutte le istruzioni dello Z-80 implicano operazioni su dati che possono essere memorizzati in registri interni alla CPU, in memoria, o che possono essere trasferiti da porte di I/O. Il termine "metodo d'indirizzamento" si riferisce al metodo tramite il quale l'istruzione accede a questi dati. I dati fanno parte dell'istruzione? L'istruzione contiene un puntatore che dice al computer dove sono i dati? L'istruzione contiene un puntatore che indica la locazione in memoria dove sono memorizzati i dati? Lo Z-80 ha in tutto dieci differenti metodi d'indirizzamento. Li prenderemo accuratamente in esame in questo capitolo e in quelli successivi.

La varietà e l'efficacia dei metodi d'indirizzamento dello Z-80 contribuiscono in larga misura ad avvantaggiare questo microprocessore a 8 bit rispetto ad altri, come l'Intel 8080A. Ovviamente, rendono più complesso il set di istruzioni dello Z-80. Comunque, potete essere certi che gli sforzi e il tempo in più trascorso ad imparare i metodi d'indirizzamento, saranno ampiamente ripagati in molte maniere. Innanzitutto, più modi esistono di estrarre e manipolare i dati, più è facile scrivere dei programmi efficienti. Secondo, la Zilog corporation ha escogitato un ottimo sistema di riferimenti incrociati fra il codice mnemonico dello Z-80 e quello esadecimale, che richiede da parte dell'utente la conoscenza e la comprensione dei dieci metodi d'indirizzamento. Altre ancora sono le ricompense che andranno a coloro che persevereranno nell'imparare i metodi d'indirizzamento, ma non staremo ad elencarle in questa sede. Vi diamo un ultimo consiglio prima di passare a presentarvi il primo gruppo di metodi d'indirizzamento: sforzatevi inizialmente di capire il significato dei metodi d'indirizzamento, piuttosto che impararne a memoria i nomi talvolta fantasiosi e non intuitivi. Il fatto di impararli a memoria verrà più tardi, con l'esperienza. In conclusione, i metodi d'indirizzamento sono importanti, e vi raccomandiamo di dedicare loro il tempo necessario per impararli.

ISTRUZIONI DI CARICAMENTO DI UN SOLO REGISTRO: METODO D'INDIRIZZAMENTO TRA REGISTRI LD d,s

Nel set di istruzioni dello Z-80, esistono 63 diverse istruzioni LD che coinvolgono solo registri singoli (non coppie).

Ogni istruzione ha il codice mnemonico LD d,s dove

d = registro destinazione
s = registro sorgente

I codici istruzione vanno esclusivamente da 40 a 7F, con la sola eccezione di 76, che è l'istruzione di Alt (HALT). Ecco l'istruzione LD sotto forma di otto bit:

0 1 D D D S S S

I valori di DDD o SSS sono i tre bit che corrispondono al codice binario a tre bit che indica il registro. Quindi:

Registro	DDD o SSS Codice binario
B	000
C	001
D	010
E	011
H	100
L	101
(HL)	110
A	111

Siamo ora pronti per dare la definizione di "metodo d'indirizzamento tra registri":

Indirizzamento tra registri	La tecnica di usare gruppi di bit all'interno del codice istruzione dello Z-80 per specificare quale registro o registri sono coinvolti nel trasferimento dati.

Ecco alcuni esempi di istruzioni appartenenti a questa classe:

Carica il registro C con il registro B	LD C,B
Carica l'accumulatore con il registro C	LD A,C
Carica il registro D con il registro E	LD D,E
Carica l'accumulatore con il registro H	LD A,H
Carica il registro L con l'accumulatore	LD L,A
Carica la locazione di memoria indirizzata dalla coppia di registri HL con l'accumulatore	LD (HL),A
Carica l'accumulatore con la locazione di memoria indirizzata dalla coppia di registri HL	LD A, (HL)

Il trasferimento dei dati fra una locazione di memoria, (HL), ed un qualunque registro, richiede ulteriori spiegazioni, e ne parleremo in un paragrafo successivo. Le istruzioni LD A,B; LD B,A; LD A,(HL); LD(HL),A sono illustrate nella Figura 6-1. Le frecce indicano la direzione del trasferimento dei dati.

CARICA UN REGISTRO IN MODO IMMEDIATO
LD r, <B2>

Il termine *"immediato"* si riferisce al metodo d'indirizzamento secondo cui i dati che devono essere caricati nel registro r sono attualmente contenuti nel secondo byte dell'istruzione stessa <B2>. Nell'istruzione di caricamento immediato di un registro, la destinazione del byte di dati è indicata dai bit indicati con 'D':

0 0 D D D 1 1 0	Byte 1
byte di dati <B2>	Byte 2

I valori di DDD sono i codici di registro già visti. Il codice mnemonico è del tipo LD D, <B2> e il numero di stati è 7, per un tempo di esecuzione di 2,8 microsecondi; fa

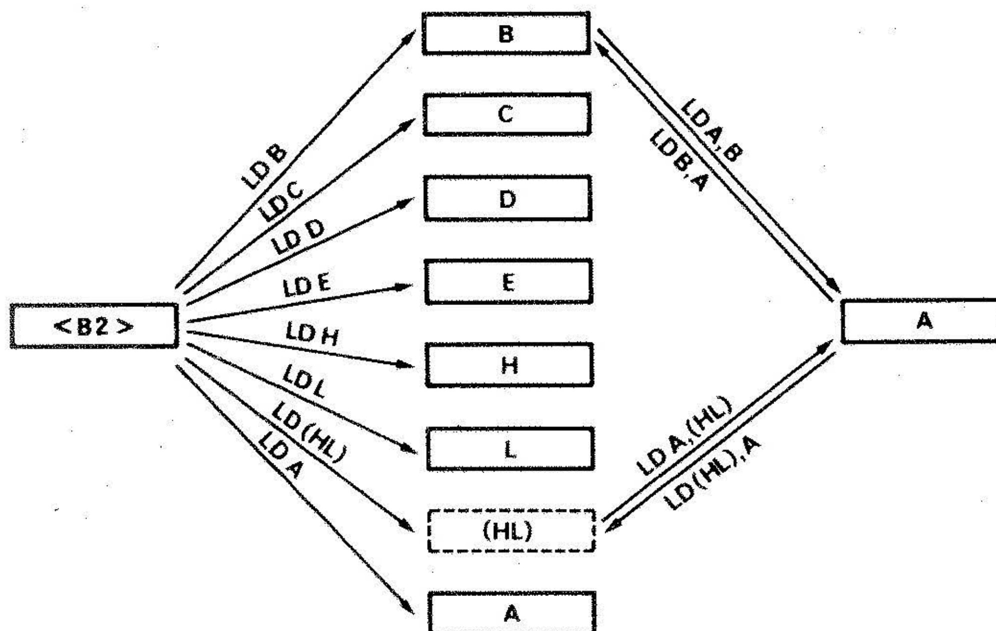


Figura 6-1

eccezione l'istruzione LD (HL), <B2>, che richiede 10 stati (4,0 microsecondi). Nella Figura 6-1 vi presentiamo le otto diverse istruzioni di caricamento di un registro in modo immediato. Il byte <B2> è il secondo byte dell'istruzione a due byte; questa informazione viene trasferita dal programma al registro designato.
Notate che *tutti i trasferimenti dei dati all'interno dello Z-80 e all'interno del Nanocomputer sono effettuati in modo parallelo; vengono trasferiti contemporaneamente otto bit di informazione.* Il trasferimento di un dato in (HL) è un'operazione che richiede ulteriori spiegazioni: ne parleremo fra poco.

CARICA L'ACCUMULATORE IN MODO INDIRETTO TRAMITE REGISTRI
LD A,(rp) LD (rp),A

L'indirizzamento indiretto tramite registri è un metodo in cui si usa una coppia di registri - BC,DE o HL - per indicare un indirizzo di memoria il cui contenuto deve essere sostituito o caricato nell'accumulatore A. Per esempio, LD A,(DE) pone il byte a otto bit, il cui indirizzo di memoria è contenuto nella coppia di registri DE, nell'accumulatore. LD (DE),A memorizza il contenuto dell'accumulatore nella locazione di memoria che viene indirizzata dal contenuto della coppia di registri DE.

LD A,(rp):	0 0 r p 1 0 1 0	oppure	LD A,(HL)	0 1 1 1 1 1 1 0
LD (rp),A:	0 0 r p 0 0 1 0	oppure	LD (HL),A	0 1 1 1 0 1 1 1

dove le coppie di registri (rp) sono codificate come indicato nella tabella seguente. La Figura 6-2 vi mostra come avviene l'indirizzamento indiretto del registro.

rp	Codice binario
BC	0 0
DE	0 1

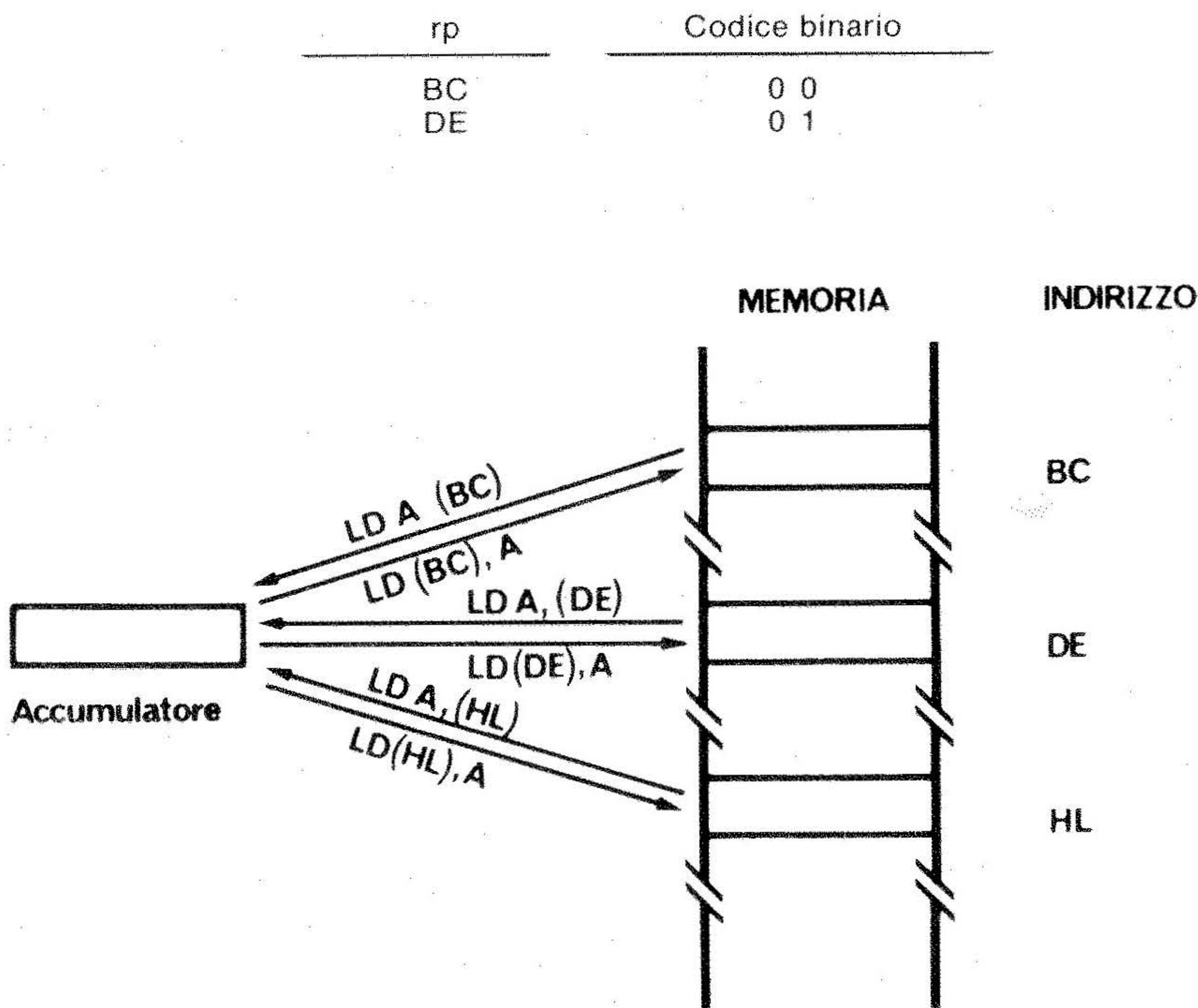


Figura 6-2

CARICA UNA COPPIA DI REGISTRI IN MODO IMMEDIATO ESTESO
LD rp, <B3> <B2>

Le istruzioni di caricamento immediato esteso appartengono al cosidetto gruppo "di caricamento a 16 bit", perchè queste istruzioni effettuano il trasferimento di due byte dell'istruzione (byte due e tre) in una coppia di registri BC,DE,HL,SP. Il termine *"immediato esteso"* si riferisce ad un altro metodo di indirizzamento dello Z-80, dove "esteso" indica un trasferimento a due byte, e "immediato" significa che i due byte di dati sono parte dell'istruzione. La Figura 6-3 mostra parecchie istruzioni di caricamento immediato esteso e come viene effettuato il trasferimento dei dati.

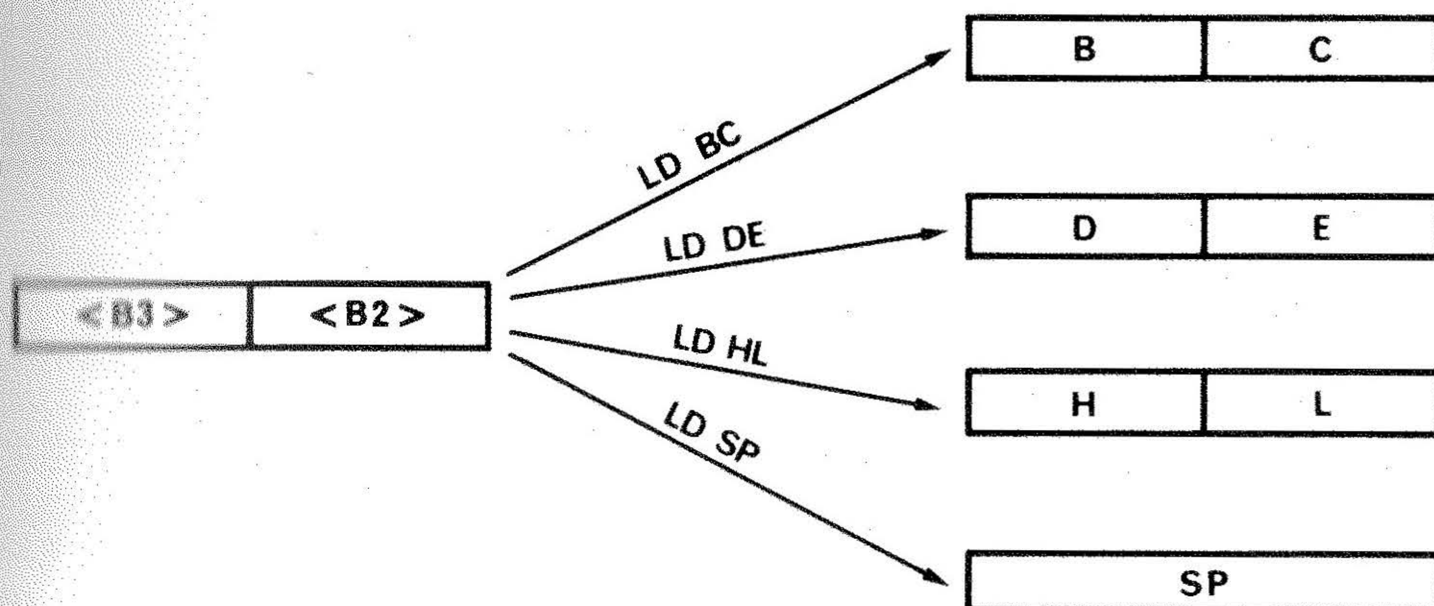


Figura 6-3

La coppia di registri di destinazione dei byte di dati è indicata dai bit contrassegnati rp nel codice operativo di queste istruzioni:

0 0 r p 0 0 0 1

dove esiste questa corrispondenza fra coppie di registri e codici a due bit:

Coppia di registri	Codice binario
BC	00
DE	01
HL	10
SP	11

CARICA UNA COPPIA DI REGISTRI IN MODO ESTESO
LD rp,(addr) LD(addr),rp

Questa istruzione usa *l'indirizzamento esteso* per trasferire due byte fra la memoria ed una coppia di registri BC, DE o HL. Indirizzamento esteso significa che due byte contenuti all'interno dell'istruzione puntano alla prima delle due locazioni di memoria che devono essere la sorgente o la destinazione del trasferimento. Per esempio, l'istruzione

LD HL,(0100H)

carica il registro L con il byte di dati contenuto nella locazione 0100 e il registro H con il byte di dati contenuto nella locazione 0101. Notate che il secondo registro della coppia viene caricato con il contenuto del "più basso" dei due indirizzi. Il codice operativo di questa istruzione è 2A, quindi LD HL,(0100H) si traduce in

2A codice operativo
00 byte dell'indirizzo di memoria LO
01 byte dell'indirizzo di memoria HI

Le operazioni di caricamento, analoghe, per le coppie di registri BC e DE hanno codici operativi a due byte. Invece di elencarvi le istruzioni e i loro codici operativi come abbiamo fatto in passato, ve li presenteremo più avanti in un formato più strutturato. Le troverete nel capitolo successivo, nella tabella del gruppo di caricamento a 16 bit.
In questo momento la cosa più importante è capire l'indirizzamento esteso.

INCREMENTA UN REGISTRO
INC r

Incrementare un registro significa aumentare il contenuto del registro di 1. L'istruzione, ad un solo byte, è semplicemente

0 0 registro 1 0 0

ed ha un codice mnemonico di INC r, dove r è il codice del registro (A, B, C ecc) su cui si vuole operare. Ad eccezione di INC (HL), l'istruzione di incremento di un registro richiede solo 5 stati, o 2,0 microsecondi, per essere eseguita.

DECREMENTA UN REGISTRO
DEC r

Decrementare un registro significa diminuire il contenuto del registro di 1. L'istruzione, ad un solo byte, è simile all' istruzione di incremento,

0 0 registro 1 0 1

ed ha un codice mnemonico di DEC r. Ad eccezione di DEC (HL), l'istruzione di decremento di un registro richiede 5 stati, o 2,0 microsecondi, per l'esecuzione. Sia le istruzioni di incremento che quelle di decremento impiegano il solito codice binario a tre bit per il registro.

SALTA SE NON ZERO
JP NZ,<B3> <B2>

Questa è la vostra prima *istruzione di Salto Condizionato,* cioè un'istruzione che è soggetta ad una condizione.

In questo caso, il salto avviene verso l'indirizzo di memoria dato da <B2> e <B3> *se il flag di zero è a livello logico 0.* Non siamo ancora preparati per parlare dei flag; per il nostro scopo basterà dire che il salto avviene solo se il risultato di un'operazione non è zero. Se il risultato dell'operazione è zero, l'istruzione JP NZ viene ignorata ed il programma passa all' istruzione successiva, saltando i tre byte dell'istruzione di salto. L'istruzione JP NZ è un'istruzione a tre byte,

1 1 0 0 0 0 1 0
byte d'indirizzo LO <B2>
byte d'indirizzo HI <B3>

che ha un tempo di esecuzione di 10 stati, o 4,0 microsecondi. Questa istruzione è molto usata per generare dei loop di ritardo (delay), di cui vi daremo un esempio in un programma di questo capitolo.

TRASFERIMENTO DI UN BLOCCO DI DATI LDD, LDI, LDDR, LDIR

Fin qui, abbiamo appreso molti modi di trasferire i dati fra i registri e le locazioni di memoria, un byte alla volta. Lo Z-80 ha quattro istruzioni molto efficaci che hanno lo scopo di facilitare lo spostamento di blocchi di dati da un gruppo di locazioni di memoria ad un altro. Prima di eseguire una qualunque di queste quattro istruzioni, un programma dello Z-80 deve "inizializzare" i registri BC,DE e HL in questo modo:

HL = indirizzo del primo byte del blocco sorgente
DE = indirizzo del primo byte del blocco destinazione
BC = numero dei byte che devono essere spostati

Ecco quanto avviene eseguendo l'istruzione LDI *("carica ed incrementa"):*

1. Il byte della locazione indirizzata dalla coppia di registri HL è caricato nella locazione indirizzata dalla coppia di registri DE.
2. Il contenuto delle coppie di registri HL e DE viene incrementato (di 1)
3. Il contenuto della coppia di registri BC viene decrementato (di 1)

Ecco che cosa accade invece eseguendo l'istruzione LDIR *("carica, incrementa e ripeti"):*

1. Il byte della locazione indicata dalla coppia di registri HL è caricato nella locazione indirizzata dalla coppia di registri DE
2. Il contenuto delle coppie di registri HL e DE è incrementato.
3. Il contenuto della coppia di registri BC è incrementato
4. Viene controllato il valore della coppia di registri BC. Se BC non è uguale a 0000, vengono ripetuti i passi 1, 2 e 3. Se BC è uguale a 0000, l'esecuzione procede con l'istruzione successiva del programma.

L'esecuzione delle istruzioni LDD e LDDR appare in sequenze di passi molto simili. L'unica differenza è che il Passo 2 *decrementa* sia HL che DE.

La Figura 6-4 di pagina seguente, mostra i registri e le locazioni di memoria prima e dopo l'esecuzione dell'istruzione LDIR.

Questa istruzione è argomento di un esperimento che eseguirete alla fine di questo capitolo.

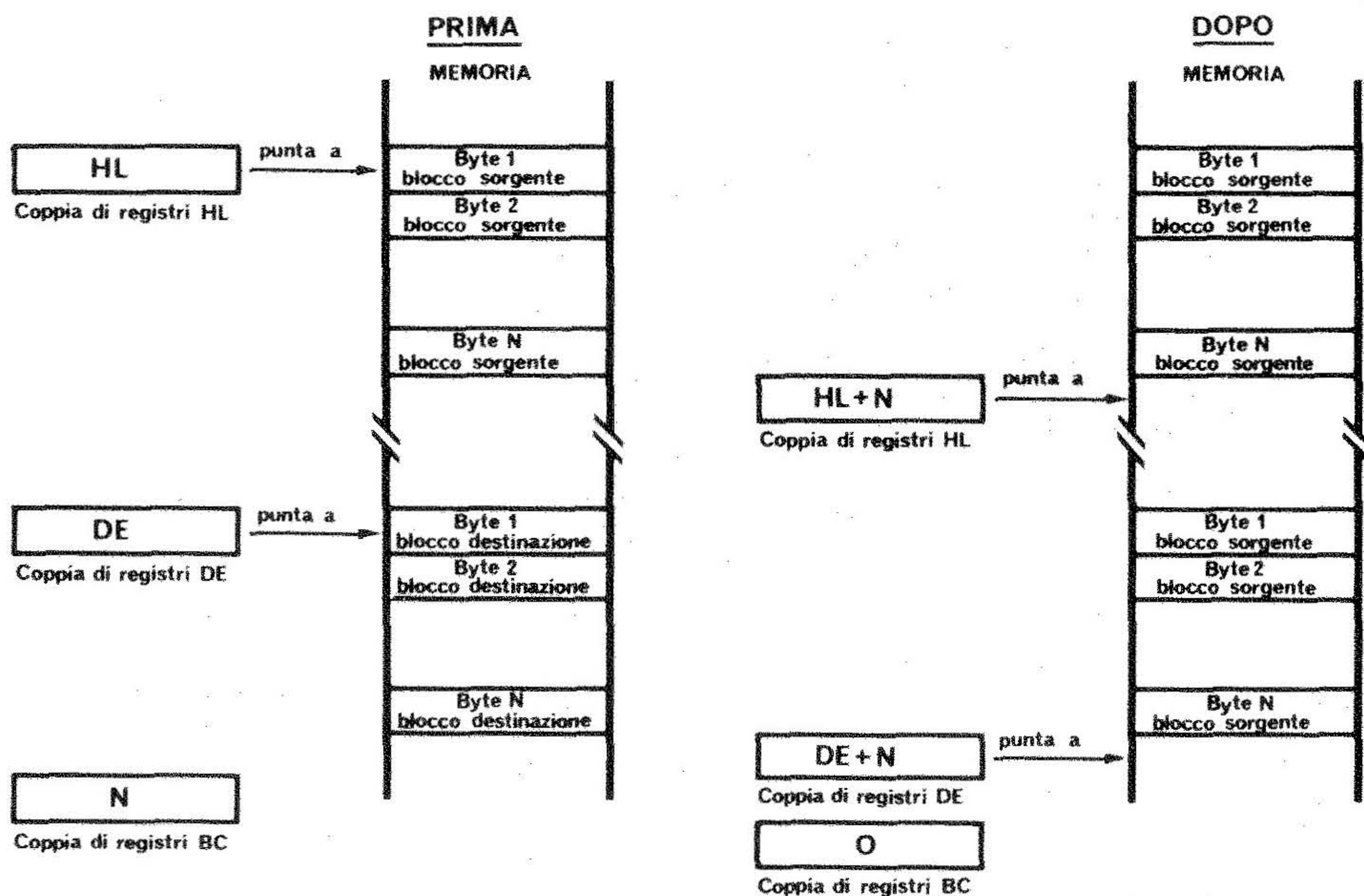


Figura 6-4

INTRODUZIONE AGLI ESPERIMENTI

Gli esperimenti che seguono dimostrano quello che avete imparato nel Capitolo 6 a proposito del trasferimento di dati fra registri e registri, fra registri e locazioni di memoria, e fra locazioni di memoria ed altre locazioni di memoria.
Gli esperimenti che eseguirete possono essere così sintetizzati:

Esperimento N.	Commenti
1	;Illustra il metodo di indirizzamento immediato e di indirizzamento ;tra registri.
2	;Illustra il metodo di indirizzamento immediato esteso, indirizzamento ;esteso e indirizzamento indiretto tramite i registri.
3	;Illustra le tecniche per realizzare dei loop di programma. ;In particolare, l'istruzione JP NZ viene usata per formare un loop di ;ritardo.
4	;Illustra l'istruzione di spostamento di blocchi dati LDDR.
5	;Illustra l'istruzione di spostamento di blocchi dati LDI. Vengono ;presentati due nuovi salti condizionati ed un'istruzione logica.
6	;Illustra il valore dell'istruzione di spostamento di blocchi dati ;facendo vedere come fa risparmiare memoria e passi di programma.

ESPERIMENTO N. 1

Scopo

Questo esperimento illustra il metodo di indirizzamento immediato e il metodo di indirizzamento tra registri.

Programma N. 7

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
0100	06 80	LD B,80H	;Indirizzamento immediato: il byte di dati 80 è ;caricato nel registro B
0102	04	INC B	;Somma uno (incrementa) al registro B
0103	48	LD C,B	;Indirizzamento tra registri; il contenuto del ;registro B è caricato nel registro C
0104	0C	INC C	;Incrementa il contenuto del registro C
0105	51	LD D,C	;Carica D con C
0106	14	INC D	;Incrementa D
0107	5A	LD E,D	;Carica E con D ;Diminuisci il contenuto del registro E di 1 ;(decremento)
0109	63	LD H,E	;Carica H con E
010A	25	DEC H	;Decrementa H
010B	6C	LD L,H	;Carica L con H
010C	2D	DEC L	;Decrementa L
010D	7D	LD A,L	;Carica A con L
010E	76	HALT	;Alt del microcomputer

Passo 1

Caricate il programma in memoria sopra riportato partendo dalla locazione 0100. Verificate che sia stato caricato correttamente.

Quante istruzioni ad un byte ci sono nel programma?
Quante istruzioni a due byte?
Quante istruzioni a tre byte?
Quante istruzioni a quattro byte?

Le vostre risposte dovrebbero dovuto essere 13, 1, 0 e 0, rispettivamente. LD B,80H è l'unica istruzione a due byte, mentre tutte le altre sono istruzioni ad un byte.

Quante istruzioni usano l'indirizzamento immediato?
Quante istruzioni usano l'indirizzamento tra registri?

Le vostre risposte avrebbero dovuto essere 1 e 6, rispettivamente.

L'istruzione LD B,80H usa l'indirizzamento immediato perchè i dati sono parte dell'istruzione (byte due). Le altre istruzioni LD, INC e DEC (tutte di un byte) usano l'indirizzamento tra i registri.

Passo 2

Analizzate il programma e dite quale sarà, secondo voi, il valore contenuto in ogni registro alla conclusione dell'esecuzione del programma. Potete scrivere le vostre previsioni nello spazio seguente:

B = ________ C = ________ D = ________ E = ________
H = ________ L = ________ A = ________

Passo 3

Verificate le vostre previsioni eseguendo il programma. Se lo eseguite con la tecnica del passo-passo, potete controllare le vostre previsioni per ogni registro man mano che viene cambiato. Se invece lo eseguite alla massima velocità, ricordatevi di premere il tasto BREAK invece di RESET, in modo che i contenuti dei registri restino intatti.
Nota: E' particolarmente interessante effettuare l'esecuzione passo a passo perchè vi permette di guardare più di un registro alla volta.
Con la luce di selezione in posizione BC, potete osservare direttamente gli effetti delle prime quattro istruzioni. Poi spostate la luce di selezione sulla posizione HL per le quattro istruzioni seguenti e in posizione AF per le istruzioni finali.

I registri dovrebbero ora contenere quanto segue:

B = 81, C = 82, D = 83, E = 82, H = 81, L = 80 e A = 80.

E' essenziale essere in grado di prevedere il contenuto dei registri che vengono utilizzati in un programma, perchè potete così scoprire le inesattezze del programma se le previsioni non risultano verificate. Vi raccomandiamo perciò di arrivare a sviluppare il più possibile questa vostra capacità.

Passo 4

Cambiate il byte di dati alla locazione 0101 in 01. Prevedete il contenuto dei registri B,C,D,E,H,L e A dopo l'esecuzione del programma così cambiato .

B = ________ C = ________ D = ________ E = ________
H = ________ L = ________ A = ________

Eseguite il programma passo-passo, guardando i registri man mano che cambiano. Al termine dell'esecuzione i registri dovrebbero contenere

B = 02, C = 03, D = 04, E = 03, H = 02, L = 01 e A = 01.

Passo 5

Cambiate il byte di dati alla locazione 0101 in FF. Cercate di prevedere quello che conteranno ora i registri B, C, D, E, H, L e A. Controllate le vostre previsioni eseguendo il programma passo a passo e guardando i registri man mano che cambiano.

B = ________ C = ________ D = ________ E = ________
H = ________ L = ________ A = ________

Abbiamo osservato che B = 00, C = 01, D = 02, E = 01, H = 00, L = FF e A = FF.
Questi valori si possono facilmente spiegare se sapete una cosa, cioè che l'addizione, sul microcomputer Z-80, è ciclica. Ovvero:

Se incrementiamo FF di uno otteniamo 00
Se decrementiamo 00 di uno otteniamo FF

Possiamo esprimere questo concetto anche in un altro modo: lo Z-80 somma in modulo 256 (base 10) o modulo 100 (base 16). Quando si verifica "un passaggio attraverso lo zero" (ad esempio quando si incrementa FF di uno), lo Z-80 rileva tale situazione mettendo a uno un flag di carry (riporto). Di questo parleremo in un capitolo successivo.

ESPERIMENTO N. 2

Scopo

Questo esperimento illustra il modo di indirizzamento esteso, il modo di indirizzamento immediato esteso, e il modo di indirizzamento indiretto tramite i registri.

Programma N. 8

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
0110	21 1C 01	LD HL,011CH	;Indirizzamento immediato esteso: H è ;caricato con 01 (HI) e L è caricato con 1C (LO)
0113	36 FF	LD (HL),FFH	;Indirizzamento indiretto tramite i registri HL ;La locazione di memoria indicata dal ;contenuto di HL è caricata con FF
0115	2C	INC L	;Incrementa il registro L in modo che HL ;indichi la locazione di memoria successiva
0116	36 EE	LD (HL),EEH	;Indirizzamento indiretto - La locazione di ;memoria indicata dal contenuto di HL è ;caricata con EE
0118	2A 1C 01	LD HL,(011CH)	;Indirizzamento esteso: il registro L è caricato ;con il contenuto della locazione di memoria ;011C e il registro H è caricato con il contenuto ;della locazione di memoria 011D
011B	76	HALT	;Alt

Passo 1

Caricate in memoria il programma sopra riportato partendo dalla locazione 0110. Verificate che sia stato caricato correttamente.

Passo 2

Esaminiamo da vicino il programma riportato al passo 1 per cercare di capire che cosa fa. Per prima cosa notate che il programma coinvolge i registri H ed L e due locazioni di memoria 011C e 011D. Il programma in sostanza carica queste due locazioni con FF ed EE, rispettivamente, e poi sposta il contenuto delle due locazioni nella coppia di registri HL. Non si tratta di un programma particolarmente entusiasmante, ma esso illustra dei particolari importanti che riguardano tre tipi di indirizzamento.

Consideriamo le due istruzioni mnemoniche:

```
LD HL,011CH
LD HL,(011CH)
```

L'unica differenza è che la seconda istruzione ha l'indirizzo scritto fra parentesi. Questa differenza è molto importante. Nella prima istruzione 011C rappresenta i *due byte* che

devono essere caricati in H ed L; nella seconda istruzione 011C è un *indirizzo.* Entrambe queste istruzioni caricano 16 bit (o due byte) di dati, ma la prima usa l'indirizzamento *immediato esteso* e la seconda usa l'indirizzamento *esteso.*
Le istruzioni LD (HL),FFH e LD (HL),EEH sono entrambe istruzioni di caricamento a 8 bit perché viene coinvolto solo un byte. Anche qui le parentesi che racchiudono HL sono importanti. Esse implicano che le istruzioni non operano su HL ma sulla locazione di memoria indicata da HL.

Passo 3

Prevedete i valori che avranno i registri e le locazioni di memoria seguenti a programma eseguito:

H = ________ (011C) = ________
L = ________ (011D) = ________

Passo 4

Eseguite il programma alla massima velocità e quindi esaminate i registri e la memoria per vedere se le vostre previsioni erano esatte.
Abbiamo osservato che H = EE, L = FF, (011C) = FF, (011D) = EE.

ESPERIMENTO N. 3

Scopo

Lo scopo di questo esperimento è illustrare come possono essere realizzati dei loop (gruppi di istruzioni che vengono ripetutamente eseguite in modo ciclico) di programma. In particolare, viene usata l'istruzione JP NZ per formare un loop di ritardo (delay).

Programma N. 9.

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
0120	0E 00		LD C,00H ;con 00	;Carica in modo immediato C
0122	0D	LOOP:	DEC C	;Decrementa C
0123	C2 22 01		JP NZ,LOOP	;Se C non è zero, ritorna a LOOP
0126	FF		RST 38H	;Se C è zero, restituisci il controllo ;al sistema operativo del ;Nanocomputer.

Programma N. 10.

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
0130	06 00		LD B,00H ;con 00	;Carica in modo immediato B
0132	0E 00	LOOP1:	LD C,00H	;Carica in modo immediato C ;con 00
0134	0D	LOOP2:	DEC C	;Decrementa C
0135	C2 34 01		JP NZ,LOOP2	;Se C non è zero, torna a LOOP2

```
0138    05          DEC B         ;Se C è zero, decrementa B
0139    C2 32 01    JP NZ,LOOP1   ;Se B non è zero, torna a LOOP1
013C    FF          RST 38H       ;Se B è zero, restituisci il controllo
                                  ;al sistema operativo del
                                  ;Nanocomputer.
```

Passo 1

Caricate e verificate il Programma N. 9.
Caricate e verificate il Programma N. 10.

Passo 2

Esaminiamo con attenzione questi due programmi per capire esattamente che cosa fanno. Prendiamo prima il Programma N. 9. Si tratta di un semplice loop di ritardo. Il registro C è inizialmente caricato con zeri, poi viene ripetutamente decrementato finchè C arriva di nuovo a 00, dove finisce il loop ed il controllo viene rimandato al sistema operativo del Nanocomputer ripartendo dalla sua prima istruzione. Il Programma N. 9 viene chiamato loop di ritardo perchè esso esegue per un certo tempo delle operazioni non efficaci rispetto all'esterno, (decrementando C) e poi si ferma. Il risultato è un ritardo; notate che il contenuto di C è zero sia all'inizio che alla fine.

Per i programmi che contengono dei loop, è spesso utile tracciare dei diagrammi di flusso (flow-chart) per avere un quadro di tutta la logica del programma.

Nella Figura 6-5 vi mostriamo il diagramma di flusso del Programma N. 9.

Vi spieghiamo a lato il significato della forma dei blocchi nel diagramma di flusso.

Notate che in questo programma sono utilizzate delle LABEL (etichette); ad alcuni statement (cioè linee del programma sorgente) sono cioè associati dei nomi di identificazione. Nel Programma N. 9, per esempio, allo statement DEC C posto alla locazione 0122 è assegnata la label LOOP. Potete riconoscere che LOOP è una label poiché è eseguita da due punti (:) e da un'istruzione.

BLOCCO DI ELABORAZIONE: esegui l'azione descritta, passa a quella successiva

BLOCCO DI ELABORAZIONE

BLOCCO DI DECISIONE: Determina la risposta alla domanda formulata ed effettua il salto di conseguenza. Ovviamente, ogni salto deve essere esplicitamente etichettato con la risposta associata

BLOCCO DI ELABORAZIONE

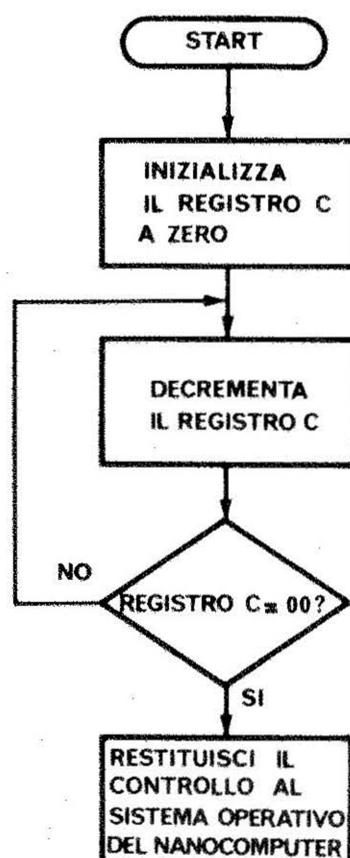


Figura 6-5

Più oltre, sempre nello stesso programma, l'istruzione JP NZ,LOOP fa riferimento alla label LOOP considerandola un sinonimo dell'indirizzo 0122. L'istruzione JP NZ,LOOP è cioè del tutto equivalente all'istruzione JP NZ, 0122H e come tale verrà convertita dal programma assembler nel codice oggetto esadecimale C2 22 01.
E' importante che vi sia chiaro che le label, presenti negli statement del codice sorgente, sono sempre tradotte in codice oggetto nei loro equivalenti esadecimali, cosicché la CPU Z80 non vede *mai* delle label.
Notate anche che il Programma N. 9 fa uso della istruzione RST 38H che non abbiamo ancora spiegato. In sintesi, questa istruzione fa si che la CPU Z80 riporti il controllo al sistema operativo del Nanocomputer. Daremo più oltre una descrizione dettagliata delle istruzioni RST.
E' opportuno fare delle osservazioni generali sui loop di programma: si può notare infatti che tutti i loop di programma contengono queste quattro componenti:

1. *Processo di inizializzazione*: variabili di conteggio, indirizzi di memoria, registri ed altre variabili necessarie vengono definite ai valori di partenza desiderati (es. LD C,00H nel Programma N. 9).

2. *Processo da ripetere:* questa parte è costituita dalle istruzioni che verranno eseguite in ogni loop. (Nota: questa componente non esiste nel Programma N. 9. Ecco perché viene chiamato loop di ritardo).

3. *Processo di controllo del loop :* vengono aggiornati le variabili di conteggio e tutti gli altri puntatori di memoria necessari o altri valori che controllano la frequenza con cui deve essere ripetuto il loop, (es. DEC C nel Programma N. 9).

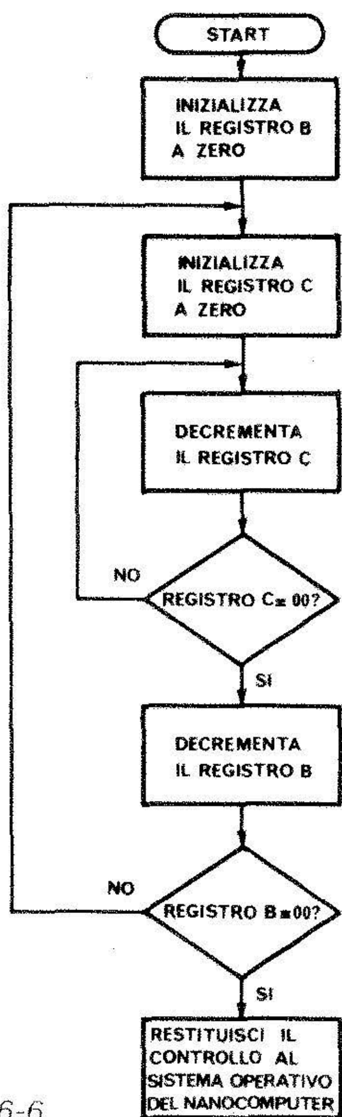


Figura 6-6

4. *Processo di conclusione del loop:* le variabili di controllo del loop vengono controllate per determinare se si è verificata una condizione di termine. L'esecuzione del loop continua o si arresta a seconda del risultato di tale controllo (es. JP NZ, LOOP nel Programma N. 9).

Analizziamo ora in modo analogo il Programma N. 10. Per prima cosa, cercate di tracciare un diagramma di flusso del Programma N. 10. Confrontate il vostro con il nostro, che vedete nella Figura 6-6. Notate che il diagramma di flusso mostra molto chiaramente la struttura logica del Programma N. 10: il Programma N. 10 è costituito da un loop interno ad un altro loop. Perchè il registro B venga decrementato una volta, il registro C deve essere prima stato decrementato di 256, passando cioè da 00 ad ancora 00.

Assicuratevi di aver capito bene quanto succede. Tutto un ciclo di decrementi (256) di C ha come risultato un solo decremento di B. Quindi quale programma, secondo voi, forma un loop di ritardo più lungo, il N. 9 o il N. 10?

Speriamo che abbiate detto il N. 10 !

Eseguite il Programma N. 9 alla massima velocità. Che cosa osservate? Eseguite il Programma N. 10 alla massima velocità. Che cosa osservate? E rispetto al Programma N. 9?

In entrambi i casi abbiamo osservato che i display del Nanocomputer si sono spenti per un istante e poi si sono riaccesi tutti con la luce di selezione sulla posizione PC. Con il Programma N. 9, dopo aver premuto GO, i display si sono accesi immediatamente, mentre con il Programma N. 10 i dislay sono stati spenti per un intervallo di tempo più lungo. Forse mezzo secondo.

Passo 4

Innanzitutto cerchiamo di scoprire come possiamo allungare l'intervallo di tempo che intercorre da quando premiamo il tasto GO a quando vediamo i display accendersi. Un modo è quello di aggiungere un altro loop, rendendo così il programma un loop interno ad un loop, a sua volta interno ad un altro loop. Questa tecnica è conosciuta come la formazione di *Loop Nidificati.* Nel Programma N. 10, il loop che decrementa il registro C è "nidificato" all'interno del loop che decrementa il registro B.

Cambiate il Programma N. 10 nel modo seguente:

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
012E	16 30	LD D,30H	;Inizializza il counter del loop più ;esterno
- - - - - - - - - lasciate invariate le locazioni 0130-013B - - - queste istruzioni formano ora - - - - - - - - - - i loop interni -			
013C	15	DEC D	;Decrementa il contatore del ;loop più esterno
013D	C2 30 01	JP NZ,0130H	;Esegui di nuovo i due loop ;interni se D non è zero
0140	FF	RST 38H	;Se D è 0, rimanda il ;controllo al sistema operativo del ;Nanocomputer

Passo 5

Disegnate un diagramma di flusso del Programma N. 10 modificato, il cui indirizzo d'inizio è 012E. Studiatelo da cima a fondo per capire la funzione di tutti e tre i loop.

Passo 6

Eseguite il programma partendo da 012E: dovreste aspettarvi un ritardo molto più lungo dal momento in cui premete GO al momento in cui il controllo ritorna al sistema operativo del Nanocomputer (cioè quando i display si riaccendono). Aspettate pazientemente, il ritardo è molto lungo. (Se dopo un minuto non è però successo niente, qualcosa non va. Premete RESET e ricontrollate che il vostro programma sia stato caricato correttamente).

Passo 7

Potete variare la durata del loop di ritardo variando il valore iniziale del registro D. Più il valore è alto, più il loop sarà lungo. Provate ad assegnare valori diversi al byte di dati nella locazione 012F per verificare tutto questo.

ESPERIMENTO N. 4

Scopo

Questo esperimento illustra l'istruzione che esegue il trasferimento di blocchi di dati LDDR.

Programma N. 11

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
0150	21 75 01	LD HL,0175H	;Specifica l'indirizzo finale ;del blocco sorgente di dati
0153	11 6F 01	LD DE,016FH	;Specifica l'indirizzo finale ;della destinazione
0156	01 05 00	LD BC,0005H	;Specifica il numero di byte ;che devono essere trasferiti
0159	ED B8	LDDR	;Sposta tutto il blocco di byte
015B	FF	RST 38H	;Riporta il controllo al ;sistema operativo del ;Nanocomputer

Passo 1

Caricate il programma in memoria partendo da 150. Verificate se lo avete caricato correttamente.

Passo 2

Esaminate attentamente il programma per scoprire esattamente che cosa fa. La sua funzione base è quella di utilizzare l'istruzione LDDR per spostare cinque byte di dati. L'istruzione LDDR è una delle numerose ed efficaci funzioni presenti nello Z-80 non implementate invece sul microprocessore Intel 8080. Di seguito riportiamo il modo in cui l'istruzione LDDR viene eseguita nel programma suddetto:

LDDR: (0175) trasferito a (016F)
(0174) trasferito a (016E)
(0173) trasferito a (016D)
(0172) trasferito a (016C)
(0171) trasferito a (016B)

Viene spostato un totale di BC=0005 byte nell'ordine in cui li abbiamo elencati.

Passo 3

Inizializzate il contenuto delle locazioni di memoria 0171-0175 come segue:

(0171) = AA
(0172) = BB
(0173) = CC
(0174) = DD
(0175) = EE

Eseguite il programma alla massima velocità; poi esaminate le locazioni di memoria 016B-016F. Scrivete di seguito quanto osservate:

(016B) =
(016C) =
(016D) =
(016E) =
(016F) =

In tali locazioni abbiamo osservato AA, BB, CC, DD, EE, rispettivamente.

Così l'istruzione LDDR provoca il trasferimento di cinque byte di memoria. Se BC fosse inizializzato a 0006 o 0003, verrebbero trasferiti rispettivamente 6 o 3 byte. Inizializzate di nuovo i byte di memoria 016B-016F introducendo, per esempio, dei valori 11 e provate le possibilità BC = 0003 e BC = 0006 cambiando il programma di conseguenza (locazioni 0157 e 0158).

Passo 4

Azzeriamo ora i 20 byte di memoria fra 0161 e 0175 compreso, apportando a tal fine alcuni cambiamenti al programma.

1. Cambiate LD DE,016FH in LD DE,0174H
2. Cambiate LD BC,0005H in LD BC,0014H
3. Introducete 00 nella locazione di memoria 0175

Eseguite il programma partendo da 0150. Esaminate le locazioni di memoria da 016A fino a 0175. Sono tutte zero?
Noi abbiamo visto che lo erano.

Passo 5

Cerchiamo di spiegare che cosa abbiamo appena fatto. Per prima cosa, ecco il codice sorgente del programma già modificato:

```
LD HL,0175H
LD DE,0174H
LD BC,0014H
LDDR
RST 38H
```

Perciò, la sequenza di trasferimenti è

(0175) → (0174)
(0174) → (0173)
(0173) → (0172)
. . .
(0162) → (0161)

E' stato trasferito un totale di 20 (base 10) o 14 (base 16) byte.
Caricando 00 nella locazione di memoria 0175 (vedi Passo 3) è iniziato un effetto "domino". Il primo trasferimento ha azzerato la locazione 0175, poi il contenuto di 0174, zero, è stato trasferito a 0173, e così via ...

Passo 6

Osservate attentamente i valori di tutte e tre le coppie di registri dopo l'esecuzione del programma:

		Le nostre osservazioni
HL =	______________	0161
DE =	______________	0160
BC =	______________	0000

HL è uguale all'indirizzo dell'ultimo byte sorgente trasferito meno uno. DE è uguale all'indirizzo dell'ultimo byte destinazione trasferito meno uno. BC = 0000.

Passo 7

Che cosa succede se eseguiamo il programma suddetto partendo da BC inizializzato a 0000? Vi sono due possibilità, che dipendono da come lo Z-80 esegue l'istruzione LDDR. Consideriamo questi due casi per uno Z-80 che ha appena incontrato un'istruzione LDDR:

CASO 1:
- PASSO 1 - trasferisci il byte di dati: (HL) a (DE)
- PASSO 2 - decrementa HL,DE e BC
- PASSO 3 - controlla se BC=0000. Se non lo è, ritorna al Passo 1, altrimenti passa all'istruzione seguente

CASO 2:
- PASSO 1 - controlla se BC=0000. Se non lo è, continua con il Passo 2, altrimenti passa all'istruzione seguente
- PASSO 2 - trasferisci il byte di dati: (HL) a (DE)
- PASSO 3 - decrementa HL,DE e BC: ritorna al Passo 1.

Se BC inizialmente non è zero, allora in entrambi i casi si producono risultati identici. Che succede se BC è inizializzato a zero? I due casi sono del tutto diversi. Nel caso 1 si tenterà di spostare 64K byte mentre nel caso 2 verranno spostati 0 byte.
Facciamo una semplice prova per vedere quale comportamento segue lo Z-80. Cambiate semplicemente l'attuale programma sostituendo il 14 nella locazione 0157 con 00. A che cosa porta questo? L'istruzione LD BC,0014H è sostituita con l'istruzione LD BC,0000H.

Passo 8

Eseguite il programma (partendo da 0150) con la tecnica del *passo-passo.* Prima guardate le tre coppie di registri che vengono inizializzate (3 passi). Ora guardate la coppia di registri BC per il Passo 4.
Che cosa avete osservato? Questa osservazione dovrebbe rispondere alla nostra domanda. Quale caso segue lo Z-80: 1 o 2?

Abbiamo osservato che BC è diventato FFFF! Quindi lo Z-80 decrementa *prima* di controllare, perciò possiamo concludere che viene seguito il caso 1 e non il caso 2. Siamo andati a scoprire una "condizione limite" dell'istruzione LDDR. Le condizioni limite legate al controllo dei loop sulla prima e/o sull'ultima ripetizione sono sempre molto critiche. Molti errori di programma nell'esecuzione dei loop sono dovuti alle condizioni limite non correttamente implementate.

Passo 9

Soffermiamoci ancora un poco sulle osservazioni ora fatte. Abbiamo appena dato inizio ad uno spostamento a blocchi di 64K byte! Non abbiamo nemmeno 64K byte di memoria. Ma, cosa più importante, ci avviamo a scrivere sopra il nostro programma. Consideriamo i seguenti trasferimenti che avranno luogo:

(0175) → (0174)
(0174) → (0173)

. . .

(0160) → (015F)
(015F) → (015E)

. . .

(015B) → (015A)
(015B) → (0159)

Con il trasferimento del contenuto della locazione di memoria 015E alla locazione 015D, stiamo incominciando ad alterare il programma in esecuzione. Lo Z-80 non ne è ancora al corrente perchè sta tranquillamente proseguendo l'esecuzione sull'istruzione LDDR alle locazioni 0159 e 0160.
Proseguite l'esecuzione passo a passo guardando il registro DE.
Quando DE=015D state iniziando a scrivere sopra (addirittura "mangiare") il vostro programma. Fin dove pensate di arrivare?
Proseguite passo-passo. Ad ogni passo il registro DE discende di un byte avvicinandosi all'istruzione LDDR. Quando smette di decrementare il registro DE?

Abbiamo osservato che ha smesso a 0159. Ovvero, una volta che il secondo byte dell'istruzione LDDR è stato cambiato, il programma è stato messo da parte. E' rimasto finchè ha potuto. Una volta che l'istruzione che stava eseguendo viene distrutta, esso non può più proseguire.

Siete stati testimoni, a condizioni molto ben controllate, del fatto che un programma si distrugge da solo. Purtroppo, questa non sarà l'ultima volta che vi succederà. Ricordatevi che questa situazione può avvenire e fate del vostro meglio per evitarla. In qualunque momento eseguiate un programma non ancora messo a punto, preparatevi al peggio copiandolo prima su di un'audiocassetta (se possibile) o comunque scrivendolo da qualche parte, perchè può scomparire quando premerete il tasto GO!

ESPERIMENTO N.5

Scopo

Questo esperimento illustra l'esecuzione dell'istruzione LDI. Vengono presentate due nuove istruzioni di salto condizionato: JP Z e JP PE, ed anche l'istruzione logica OR A.

Programma N. 12

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
0180	21 A0 01		LD HL,01A0H	;Specifica l'indirizzo d'inizio del ;blocco sorgente di dati
0183	11 C0 01		LD DE,01C0H	;Specifica l'indirizzo d'inizio del ;blocco destinazione dei dati
0186	01 10 00		LD BC,0010H	;Specifica il numero massimo di ;byte che devono essere spostati
0189	7E	LOOP:	LD A,(HL)	;Carica il prossimo byte sorgente ;che dev'essere trasferito nel ;registro A
018A	B7		OR A	;Porta il flag di zero al livello logico ;1 se A è 0
018B	CA 93 01		JP Z,QUIT	;Se A è zero, salta alla fine del ;programma
018E	ED A0		LDI	;Trasferisci il byte diverso da 0
0190	EA 89 01		JP PE,LOOP	;Salta all'indietro per trasferire un ;altro byte se BC non è 0000
0193	FF	QUIT:	RST 38H	;Rimanda il controllo al sistema ;operativo del Nanocomputer

Passo 1

Caricate il programma partendo dall'indirizzo 0180. Verificate se lo avete caricato correttamente.

Passo 2

Descriviamo per prima cosa le nuove istruzioni che compaiono in questo programma:

Codice oggetto	Codice mnemonico	Operazione
B7	OR A	Esegue un OR logico dell'accumulatore con se stesso, bit per bit. Il flag di zero è posto a 1 se A è zero, altrimenti il flag di zero è posto a zero. Vedi il capitolo sulle operazioni logiche, dove viene illustrata per esteso questa istruzione
CA <B2> <B3>	;JP Z,<B3><B2>	Salto condizionato; salta all'indirizzo dato da <B3> <B2> se il flag di zero è 1.
EA <B2> <B3>	JP PE,<B3><B2>	Salto condizionato: Salta all'indirizzo dato da <B3><B2> se il flag di parità è a 1

Entrambi i salti condizionati che abbiamo descritto sono molto simili all'istruzione JP NZ. L'unica differenza è la condizione che viene esaminata prima di decidere se saltare o no. Queste condizioni coinvolgono sempre i flag, di cui parleremo dettagliatamente più avanti. Per ora, per capire il programma di questo esperimento vi è sufficiente sapere quanto detto.

Passo 3

Esaminiamo ora il programma nell'insieme, per capire esattamente che cosa fa. Le prime tre istruzioni inizializzano le coppie di registri HL,DE e BC che verranno utilizzate dall'istruzione LDI. Le due istruzioni seguenti servono a determinare se il byte che deve essere trasferito è 00: il byte (indicato da HL) è caricato nell'accumulatore ed esegue quindi un OR con se stesso. L'unico caso in cui il risultato di questa operazione di OR è zero, è che A stesso sia zero. Perciò OR A porta il flag di zero a 1 solo se A è zero. L'istruzione seguente, il salto condizionato JP Z, esamina il flag di zero; se esso è a 1, allora A è zero cioé il byte da trasferire è 00; viene così effettuato il salto allo statement QUIT, che rimanda il controllo al sistema operativo del Nanocomputer.
Al contrario, il salto condizionato alla locazione 018B non viene effettuato se il byte da trasferire non è zero. Viene così eseguita l'istruzione LDI (cioé, il byte viene trasferito e vengono incrementati HL,DE, mentre BC è decrementato). Un particolare spesso ignorato ma cruciale a proposito delle istruzioni LDI e LDD è che, finchè BC non è zero, il flag di parità viene posto a 1. Quindi, quando BC è decrementato, viene fatto un controllo e il flag di parità è aggiornato di conseguenza. Perciò il salto condizionato JP PE controlla il flag di parità. Se quest'ultimo è a 1, BC non è zero, e il ciclo ricomincia per arrivare a determinare se il byte da trasferire successivamente è zero. Se il flag di parità è a 0, tutti i byte sono stati trasferiti, per cui il controllo viene rimandato al sistema operativo del Nanocomputer, cioé il salto dell'istruzione JP PE non viene eseguito.
Questo paragrafo è stato finora una descrizione in italiano, assai complicata, di un programma. Probabilmente, siete in grado di apprezzare la frase "una figura vale mille parole", quando fate il confronto con il diagramma di flusso della Figura 6-7. Per ora, dovrebbe essere chiaro che il programma trasferisce un blocco di memoria lungo al massimo 16 byte. Il primo byte zero nel blocco sorgente pone fine al traferimento.

Passo 4

Inizializzate il blocco di memoria di 16 byte, che parte da 01A0, riempiendolo con valori diversi da zero, ad esempio 11. Eseguite il programma passo a passo e guardate le coppie di registri BC,DE e HL che cambiano. Quali sono i valori finali di questi registri dopo che il controllo è stato riportato al sistema operativo del Nanocomputer?

HL=
DE=
BC=

Noi abbiamo osservato che HL=01B0, DE=01D0 e BC=0000.

Passo 5

Inizializzate il blocco di memoria a 16 bit, che inizia da 01A0, in questo modo:

01A0=10
01A1=0F
01A2=0E
01A3=00
da 01A4 a 01AF=FF

Eseguite di nuovo il programma. Guardate la coppia di registri BC e il registro A. Che cosa succede a BC dopo che il registro A è stato caricato con 00? Quali erano i valori

finali dei registri HL,DE e BC?

Avreste dovuto osservare che il registro BC è rimasto a 000D.
I valori finali di HL e DE sono 01A3 e 01C3, rispettivamente.
Sono stati trasferiti solo tre byte, secondo quanto avevamo appunto detto.

Notate che questo programma non può essere implementato con l'istruzione LDIR perchè sono necessarie delle manipolazioni dei dati fra un trasferimento e l'altro.

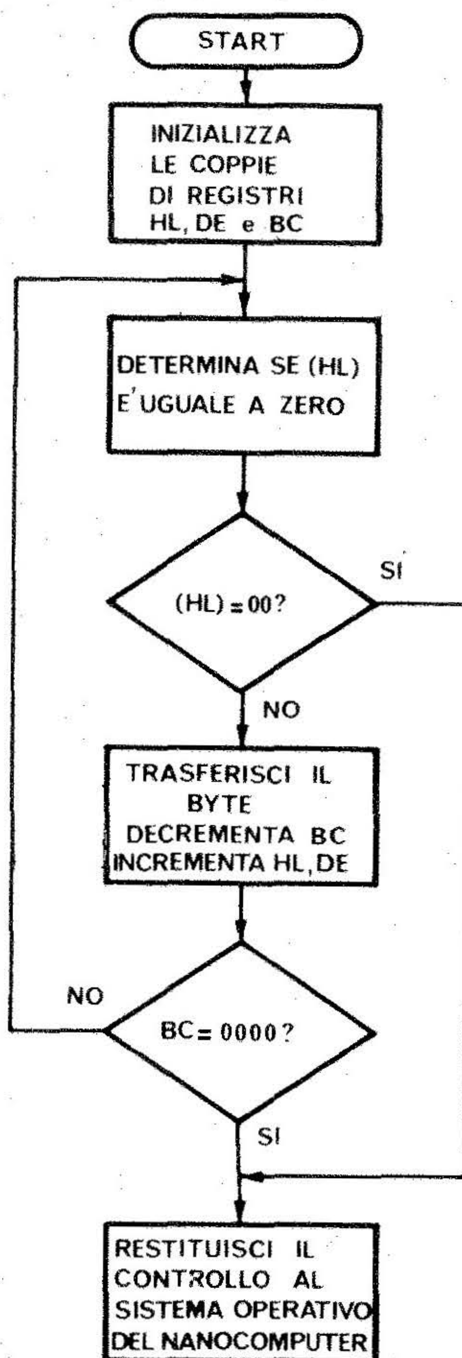


Figura 6-7

ESPERIMENTO N. 6

Scopo

Questo esperimento mostra l'importanza dell'istruzione di trasferimento di blocchi, facendovi vedere come può farvi risparmiare memoria e passi di programma.

Programma N. 13: con LDIR

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
01D0	21 00 02	LD HL,0200H	;Inizializza le tre coppie di registri
01D3	11 01 02	LD DE,0201H	;utilizzate nello spostamento dei
01D6	01 64 00	LD BC,0064H	;blocchi per specificare sorgente, ;destinazione e numero dei byte
01D9	ED B0	LDIR	;Sposta il blocco di dati
01DB	FF	RST 38H	;Trasferisci il controllo al sistema ;operativo del Nanocomputer

Programma N. 14: senza LDIR

01D0	21 00 02		LD HL,0200H	;Come sopra
01D3	11 01 02		LD DE,0201H	;Come sopra
01D6	01 64 00		LD BC,0064H	;Come sopra
01D9	7E	LOOP:	LD A,(HL)	;Carica nell'accumulatore un byte ;dal blocco sorgente
01DA	12		LD (DE),A	;Memorizza alla destinazione (DE)
01DB	23		INC HL	;Aggiorna HL: incremento su 16 bit
01DC	13		INC DE	;Aggiorna DE: incremento su 16 bit
01DD	0B		DEC BC	;Aggiorna BC: decremento su 16 bit
01DE	78		LD A,B	;Controlla se BC = 0000
01DF	B1		OR C	;Si tratta di un particolare trucco ;che vale la pena di ricordare. ;Ne parliamo nel Capitolo 9
01E0	C2 D9 01		JP NZ,LOOP	;Se BC non è zero, trasferisci un ;altro byte
01E3	FF		RST 38H	;Altrimenti riporta il controllo ;al sistema operativo del ;Nanocomputer

Passo 1

Questo esperimento serve a mostrarvi l'incredibile risparmio a cui possono portare LDIR e le altre istruzioni di spostamento a blocchi, per i programmi che spostano dati. Entrambi i programmi spostano un blocco di 100 byte di memoria consecutivi. Tuttavia, il Programma N. 13 impiega 12 byte di memoria e 2136 stati della CPU o (2136 × 0,000004) = 0,00838 secondi per l'esecuzione. Il programma N. 14 impiega 20 byte di memoria e 5043 stati o (5043 × 0,000004) = 0,02 secondi per l'esecuzione ... più del doppio! (In un altro esperimento, parleremo di come di possono calcolare questi tempi di esecuzione). Notate che la discrepanza aumenta con l'aumentare del numero di byte da trasferire.

Il motivo per cui questo confronto è interessante sta nel fatto che il microprocessore 8080 non ha istruzioni di trasferimento di blocchi di dati. Perciò su di un 8080, la procedura per spostare i blocchi di dati deve essere quella mostrata con il Programma N. 14.

Passo 2

Notate che vi sono molte istruzioni nuove presenti nel Programma N. 14. Ne rimandiamo la spiegazione dettagliata più avanti.
Per il momento ci interessa solo illustrarvi l'utilità e l'efficienza dell'istruzione LDIR e delle altre istruzioni che operano su blocchi.

Passo 3

Caricate ed eseguite ognuno dei programmi suddetti e verificate che il Programma N. 13 compie esattamente la stessa funzione del Programma N. 14.

CAPITOLO 7

METODI DI INDIRIZZAMENTO DELLO Z-80

INTRODUZIONE

Questo capitolo prosegue nella descrizione dei metodi d'indirizzamento dello Z-80, iniziata nel Capitolo 6. In particolare, prenderemo in considerazione le possibilità particolarmente importanti dell'indirizzamento indicizzato. Dato che è necessaria una conoscenza dell'aritmetica complemento a due per capire l'indirizzamento indicizzato, abbiamo incluso un paragrafo su questo argomento. Alla fine del Capitolo, vi presentiamo, sotto forma di tabella, i codici mnemonici delle istruzioni ed i codici operativi ad essi associati. Questo metodo venne suggerito per la prima volta dalla Zilog Corporation nel suo Manuale Tecnico sulla CPU dello Z-80, e noi lo abbiamo trovato molto utile.

OBIETTIVI

Alla fine di questo capitolo, sarete in grado di:

- Definire la rappresentazione binaria in complemento a due per qualsiasi numero
- Usare l'aritmetica complemento a due per eseguire le operazioni che utilizzano l'indirizzamento indicizzato
- Spiegare tutti i metodi d'indirizzamento dello Z-80 e fare degli esempi di istruzioni per ognuno di essi
- Dare la definizione di "stack" e delle operazioni ad esso associate (PUSH e POP)
- Spiegare ed usare le istruzioni di scambio
- Capire ed usare le tabelle istruzioni della Zilog per i seguenti gruppi di istruzioni:

  caricamenti ad 8 bit
  caricamenti a 16 bit
  trasferimento di blocchi di dati
  scambi di contenuti

CHE COSA E' UN METODO DI INDIRIZZAMENTO?

Il concetto di metodo d'indirizzamento è stato introdotto nel capitolo precedente. Per completarlo, ne daremo ora una definizione formale.

Metodo di indirizzamento — La tecnica mediante la quale un'istruzione fa riferimento ai dati sui quali opererà. Le istruzioni dello Z-80 dispongono in totale di dieci metodi d'indirizzamento, con alcune istruzioni che usano due metodi contemporaneamente per accedere ai dati su cui operare.

Nel Capitolo 6 abbiamo descritto i metodi d'indirizzamento tra registri, indiretto tramite registri, immediato, immediato esteso ed esteso. Gli altri metodi d'indirizzamento sono: indirizzamento modificato in pagina zero, relativo, indicizzato, implicito e su singoli bit. Gli indirizzamenti indicizzato e relativo forniscono maggiori possibilità al programmatore dello Z-80 e richiedono una buona conoscenza dell'aritmetica binaria complemento a due, che trattiamo nel paragrafo seguente.

RAPPRESENTAZIONE BINARIA COMPLEMENTO A DUE

Nel Capitolo 1, abbiamo definito un codice digitale come un sistema di simboli che rappresentano dei dati in un modo utile ai computer o ad altri circuiti digitali. La rappresentazione complemento a due è un modo di codificare gli interi, ed è molto simile alla codifica binaria. La differenza sta nel fatto che sia gli interi negativi che quelli positivi possono essere codificati usando la rappresentazione complemento a due. Inoltre, la rappresentazione complemento a due facilita notevolmente l'implementazione della addizione e della sottrazione da parte dei circuiti digitali.
Nella breve tabella che segue, vi mostriamo sia i numeri decimali positivi che quelli negativi, e la rappresentazione complemento a due su quattro bit, ad essi associata. Dobbiamo sempre specificare il numero di bit che si usano per rappresentare dei dati in complemento a due (per motivi che, più avanti, vi risulteranno ovvii).

Numero decimale	Rappresentazione complemento a due con quattro bit
7	0 1 1 1
6	0 1 1 0
5	0 1 0 1
4	0 1 0 0
3	0 0 1 1
2	0 0 1 0
1	0 0 0 1
0	0 0 0 0
—1	1 1 1 1
—2	1 1 1 0
—3	1 1 0 1
—4	1 1 0 0
—5	1 0 1 1
—6	1 0 1 0
—7	1 0 0 1
—8	1 0 0 0

Possiamo fare molte osservazioni:

1. La normale rappresentazione binaria con quattro bit ci permette di rappresentare i numeri decimali da 0 a 15.
   La codifica complemento a due con quattro bit permette di rappresentare gli interi fra −8 e +7, metà dei codici sono positivi e metà sono negativi. Perciò il complemento a due con n-bit codifica i numeri fra −2 * * (N-1) e +2 * * (N-1) − 1. (Nota: con a * * b si indica l'espressione a^b)

2. I numeri positivi hanno tutti codici complemento a due con il primo bit uguale a zero, mentre i codici dei numeri negativi iniziano con 1. Quindi, dato un numero complemento a due con quattro bit, è facile determinare se il numero è positivo o negativo .... basta guardare il primo bit (più significativo). Questo vale anche per tutti i numeri complemento a due con qualunque numero di bit.

3. Un codice complemento a due di un numero positivo decimale è identico al suo codice binario.

4. Mentre − 8 ha una rappresentazione complemento a due con quattro bit, + 8 non ce l'ha. Nella rappresentazione con n-bit, − 2 * * (N-1) è rappresentabile, ma + 2 * * (N-1) non lo è.

5. Il complemento a due di 0001 è 1111, di 0101 è 1011, di 1010 è 0110. Ovvero, dire che i numeri complemento a due "si complementano a vicenda" significa che essi rappresentano ognuno il negativo decimale dell'altro; o anche che la loro somma è zero. Vediamo se questo è vero:

0 0 0 1	1 0 1 0	0 1 0 1
+ 1 1 1 1	+ 0 1 1 0	+ 1 0 1 1
1 0 0 0 0	1 0 0 0 0	1 0 0 0 0

Eseguendo tali addizioni binarie, otteniamo qualcosa che non è zero. Ma ricordate, stiamo usando solo rappresentazioni con QUATTRO bit! Quindi, dal momento che il riporto generato dal quarto bit viene a formare un quinto bit, siamo usciti dal campo a nostra disposizione!

L'ultima osservazione è particolarmente importante perché ci mostra che cos'è in sostanza la codifica complemento a due. Essa facilita l'addizione di interi, ed anche la sottrazione, perché sottrarre un numero equivale ad addizionare il suo complemento a due. Dato un numero binario a n-bit, come si fa a trovare il suo complemento a due? Ve lo dimostriamo facendovi un esempio con quattro bit. Prendiamo il numero 0001. Per determinare il suo complemento a due, prima cambiate tutti i bit che sono a 0 in 1 e tutti i bit che sono a 1 in 0 (per quanto riguarda il nostro esempio, il risultato è 1110); quindi sommate 0001 per ottenere 1111.

Controllate sulla tabella per vedere se è giusto. Ecco alcuni esempi.

ESEMPIO 1. Trovate il complemento a due di 1010.

Passo 1:	0101
Passo 2:	+ 0001
Risposta:	0110

ESEMPIO 2. Trovate il complemento a due di 0000.

Passo 1:	1111
Passo 2:	+ 0001
Risposta:	0000

Dato che − 0 = 0 questo non ci sorprende!

ESEMPIO 3. Trovate il complemento a due di 1000

Passo 1:	0111
Passo 2:	+ 0001
Risposta:	1000

Notate che il primo bit è *uno!*

ATTENZIONE! Il complemento a due di un numero negativo dovrebbe essere positivo. Abbiamo pero messo in evidenza che − 8 (la cui rappresentazione in complemento a due è 1000) non ha complemento a due. La ragione è che il suo complemento a due è + 8, numero che non ha una rappresentazione in complemento a due su quattro bit. Tutti gli altri numeri in complemento a due su quattro bit (fra − 7 e + 7) hanno complementi su quattro bit.

Prendiamo in considerazione alcune rappresentazioni in complemento a due su otto bit.

ESEMPIO 4: Qual è l'intero positivo più grande che può essere rappresentato con un codice complemento a due su otto bit?

Risposta: Un numero positivo deve iniziare con 0, quindi il più grande numero intero è 0 1 1 1 1 1 1 1 = 127 (base 10)

Qual è l'intero negativo più grande (più grande in valore assoluto) che si può rappresentare con un codice complemento a due di otto bit? E' − 127?
Qual è la rappresentazione in complemento a due di − 127? Per illustrare tutto ciò, ci occorre solo formare il complemento a due di 0 1 1 1 1 1 1 1:

Passo 1:	1 0 0 0 0 0 0 0
Passo 2:	+0 0 0 0 0 0 0 1
Risposta:	1 0 0 0 0 0 0 1

Esiste un numero valore assoluto ancora più grande di questo. Infatti è
1 0 0 0 0 0 0 0
che è la rappresentazione in complemento a due di − 128.

Quindi i codici complemento a due su otto bit comprendono gli interi fra − 128 e + 127. Dato un codice complemento a due su otto bit, come si fa a determinare il suo equivalente decimale? Ecco alcuni esempi.

ESEMPIO 5: A quale numero decimale corrispondono i seguenti numeri in complemento a due su otto bit?

a. 00001100
b. 01100001
c. 10001111
d. 11100001

a. Dato che il codice inizia con uno zero, esso rappresenta un intero positivo e noi interpretiamo il codice complemento a due come se fosse codice binario. Perciò, la risposta è 12 (base 10).

b. Abbiamo ancora un intero positivo, per cui interpretiamo il codice binario nel solito modo e otteniamo 97 (base 10).

c. Qui abbiamo un intero negativo. Per determinare quale, formate il suo complemento a due e decodificatelo:

Il complemento a due di 10001111 è 01110001.
01110001 è la rappresentazione in complemento a due di 113 (base 10). Quindi 10001111 è la rappresentazione in complemento a due di − 113 (base 10).

d. Abbiamo un altro numero negativo, per cui seguiamo lo stesso procedimento adottato al punto c.

Passo 1: Trovare il complemento a due di 11100001, che è 00011111.
Passo 2: Identificare 00011111 come rappresentazione in complemento a due di 31 (base 10).
Passo 3: Quindi 11100001 è la rappresentazione in complemento a due di − 31 (base 10).

E dovendo fare l'operazione inversa? Cioé, dato un intero decimale compreso fra − 128 e + 127, come troviamo la sua rappresentazione complemento a due? Si adottano le stesse tecniche di base, come vedrete nell'esempio seguente.

ESEMPIO 6: Trovate la rappresentazione complemento a due dei seguenti numeri decimali:

a. 100 b. − 13

a. 100 è positivo, perciò tutto ciò che ci serve è trovare il suo codice binario, che identifichiamo subito con 01100100.
b. Questa volta abbiamo un numero negativo, quindi applichiamo la vecchia regola "complementa e codifica il positivo": in particolare troviamo la rappresentazione in complemento a due di + 13 e poi complementiamo a due tale valore!
La rappresentazione in complemento a due di + 13 = la rappresentazione binaria di + 13 = 00001101.

Quindi, la rappresentazione in complemento a due di − 13 = 11110011.

L'esempio 7 e la spiegazione che lo segue sono per coloro che desiderano sapere qualcosa sulla teoria della rappresentazione in complemento a due. Vorremmo sottolineare che non sarà necessario aver capito l'esempio 7 per essere in grado di utilizzare tutta la notevole capacità di indirizzamento indicizzato e di indirizzamento relativo dello Z-80. Comunque, parlando dell'addizione e sottrazione in complemento a due nei paragrafi successivi, faremo riferimento all'espressione (2 * * n) —x, per indicare che alcune delle regole (apparentemente) arbitrarie che stabiliremo hanno giustificazioni matematiche.
Fin qui, non vi abbiamo dato nessuna ragione per cui abbia senso rappresentare numeri positivi con il codice binario standard e numeri negativi con un codice "pazzo", risultato di un'operazione a due passi. L'operazione a due passi, che cambia gli zeri in uni e gli uni in zeri e poi somma uno, non è scesa per caso dal cielo. Questa procedura vi serve per trovare la rappresentazione binaria di (2 * * n) — x, dove x è l'intero positivo di partenza espresso in binario (n è il numero dei bit nella rappresentazione binaria). Controlliamo questa affermazione con un esempio:

*ESEMPIO 7: Sappiamo che 100 (base 10) ha una rappresentazione in complemento a due pari a 01100100 (vedi Esempio 6). Troviamo il complemento a due di 100 (base 10) usando l'espressione (2** n) —x.*

(2* * n) —x = (2* * 8) —100 = 156 (base 10)

La rappresentazione binaria di 156 è 10011100.
Il fatto di trovare —100 usando il metodo dei due passi porta ad ottenere lo stesso numero binario. Dovreste controllarlo da soli.

NOTA: se ripensate ai risultati quando sommate un numero al suo complemento a due ricorderete che abbiamo sempre ottenuto un 1 seguito da n zeri, dove n era il numero dei bit della rappresentazione. Naturalmente, 1 seguito da n zeri è la rappresentazione binaria di 2 * * n.
Quindi tutto quello che stavamo facendo era sommare x e (2 * * n) − x per ottenere 2 * * !

ADDIZIONE E SOTTRAZIONE IN COMPLEMENTO A DUE

Affrontiamo prima il problema dell'addizione. In effetti, quando siete in grado di sommare due numeri in complemento a 2, avrete risolto anche il problema della sottrazione. Perchè? Perchè qualunque problema di sottrazione (x-y) può essere ridotto ad un problema di addizione (x + (—y)).
Trovate il complemento a due di y, sommatelo a x, ed avrete eseguito la sottrazione. L'addizione di numeri complemento a due viene eseguita esattamente come se i numeri avessero rappresentazioni binarie. Questo è un grosso vantaggio della regola del complemento a due.

ESEMPIO 8:

a.

```
 0 0 0 0 0 1 1 1     (+7)
+0 0 0 0 0 0 1 0     (+2)
 ----------------------------
 0 0 0 0 1 0 0 1     (+9)
```

b.

```
 1 1 1 1 1 1 0 0     (—4)
+0 0 0 0 0 0 1 1     (+3)
 ----------------------------
 1 1 1 1 1 1 1 1     (—1)
```

c.

```
 1 1 1 1 1 0 0 1     (—7)
+1 1 1 1 0 0 1 1     (—13)
 ----------------------------
 1 1 1 0 1 1 0 0     (—20)
```

d.

```
 0 1 1 0 0 0 0 0     (+96)
+0 1 0 1 0 0 0 0     (+82)
 ----------------------------
 1 0 1 1 0 0 0 0     PROBLEMA! La somma
                     di due numeri positivi dà
                     un numero negativo!
```

e.

```
 1 0 1 1 1 0 0 1     (—71)
 1 0 1 1 1 0 0 0     (—72)
 ----------------------------
 0 1 1 1 0 0 0 1     PROBLEMA! La somma
                     di due numeri negativi
                     dà un numero positivo!
```

Nelle ultime due addizioni (d ed e) abbiamo avuto dei problemi. Quanto è successo si chiama OVERFLOW. Come ricorderete, i numeri complemento a due con otto bit vanno da —128 a + 127. quando sommiamo 96 e 82 in d, e —71 e —72 in e, le somme ottenute sono, rispettivamente , 178 e —143. Questi sono numeri che vanno al di là dei limiti di – 128 e + 127. Questo fenomeno, chiamato OVERFLOW (cioé trabocco, superamento), si può verificare ogni qualvolta i numeri vengono rappresentati con codici con una lunghezza di bit fissata. Di solito, trattare l'overflow significa scoprire quando si verifica e saltare ad un set di istruzioni che stampano un messaggio di errore. Quindi la domanda è: come scoprire l'overflow? Per alcuni codici, si tratta di un problema non facile. Fortunatamente, una delle capacità della rappresentazione complemento a due, è la facilità di rivelare l'overflow. Se la somma di due numeri positivi dà un numero negativo, o se la somma di due numeri negativi dà un numero positivo, allora e solo allora si verifica l'overflow. Il controllo consiste semplicemente nel controllare il primo bit (il bit più significativo) di ogni addendo e della somma. Lo Z-80 setta inoltre un bit (il bit P/V) nel suo REGISTRO DEI FLAG, se l'addizione del complemento a due genera un overflow. Del REGISTRO DEI FLAG parleremo per esteso in un altro capitolo.

Come vi abbiamo accennato in precedenza, la sottrazione viene eseguita complementando e sommando il dato che deve essere sottratto.

Con questo, si conclude la nostra discussione sulla rappresentazione in complemento a due.

METODI D'INDIRIZZAMENTO DELLO Z-80

Nei prossimi paragrafi parleremo delle eccellenti capacità d'indirizzamento dello Z-80. Sono proprio i dieci metodi d'indirizzamento dello Z-80 che contribuiscono in gran parte a renderlo superiore al microprocessore Intel 8080 per quanto riguarda la ricchezza del set di istruzioni. Per ogni metodo d'indirizzamento, vi diamo una spiegazione esauriente che comprende definizioni ed esempi. Per riprendere quello che abbiamo già detto nel Capitolo 6, leggete quanto segue facendo particolarmente attenzione a quello che il metodo d'indirizzamento fa, a come si confronta con gli altri metodi, e soprattutto alle notazioni usate nel codice mnemonico per ogni metodo. Lo sforzo che farete in questa parte del capitolo vi permetterà di leggere e capire le tabelle delle istruzioni Z80, che sono essenziali per poter proseguire con questo libro. Alla fine di questo capitolo vi proporremo parecchi esercizi che vi aiuteranno a rafforzare la vostra conoscenza di questi importanti concetti.

INDIRIZZAMENTO TRA REGISTRI

L'indirizzamento tra registri si verifica quando il codice operativo di un'istruzione contiene informazioni che specificano quale registro (o registri) è coinvolto nell'esecuzione dell'istruzione. I codici operativi che contengono i codici dei registri a tre bit elencati nel Capitolo 6 sono esempi di questo tipo di indirizzamento. Consideriamo l'istruzione:

LD A,B

il cui codice operativo è 0 1 1 1 1 0 0 0 o 78 esadecimale

(1 1 1 = A, 0 0 0 = B)

L'indirizzamento tra registri viene utilizzato due volte in questa istruzione, per il registro A e per il registro B.

INDIRIZZAMENTO IMMEDIATO

Il metodo *d'indirizzamento immediato* si usa con le istruzioni a più byte che contengono il byte di dati a otto bit su cui bisogna operare. La seguente istruzione di caricamento usa l'indirizzamento immediato:

LD C,03H

il cui codice esadecimale associato è: 0E 03. Al termine dell'esecuzione di questa istruzione il registro C della CPU conterà il valore esadecimale 03. (Questa istruzione utilizza qualche altro metodo d'indirizzamento? Il codice operativo indica che il registro C deve essere caricato, perciò è utilizzato anche il metodo di indirizzamento tra registri).

INDIRIZZAMENTO IMMEDIATO ESTESO

Questo metodo d'indirizzamento richiede che l'istruzione fornisca due byte di dati in modo immediato dopo il codice operativo, anzichè un byte richiesto dall'indirizzamento immediato. *L'indirizzamento immediato esteso,* quindi, "estende" il metodo dell'indirizzamento immediato.
Chiaramente, il codice macchina per qualunque istruzione che utilizzi questo metodo d'indirizzamento è lungo almeno tre byte, con un byte per il codice operativo e due byte per i dati.

L'istruzione LD BC,0421H

il cui codice esadecimale è 01 21 04, usa l'indirizzamento immediato esteso. Badate bene all'ordine in cui i byte di dati compaiono nel codice macchina per questa istruzione. Il byte per il registro C (21) è il byte LO e, pertanto, viene per primo. Questo vale per tutti i caricamenti delle coppie di registri.

INDIRIZZAMENTO INDIRETTO TRAMITE REGISTRI

Abbiamo visto dei metodi per indirizzare i registri e dei metodi in cui i dati sono parte dell'istruzione. Il metodo *d'indirizzamento indiretto tramite registri* fa uso di una coppia di registri per indicare dove risiedono i dati in memoria. Cioè, la coppia di registri contiene l'indirizzo del dato (byte) che l'istruzione richiede. Un'istruzione che usa questo tipo d'indirizzamento è

LD A,(HL)

il cui codice esadecimale è 7E. Per evidenziare che il contenuto della coppia di registri HL deve essere usato come puntatore della memoria, HL è racchiuso fra parentesi. Questa regola è standard per l'indirizzamento indiretto. Per alcune istruzioni, l'indirizzamento indiretto è utilizzato per accedere a due byte di dato. In questi casi, il contenuto della coppia di registri specifica il byte LO e il contenuto della coppia di registri più uno punta al byte HI. Per esempio l'istruzione

POP BC

il cui codice esadecimale è C1, carica (SP) in C e (SP+1) in B.

INDIRIZZAMENTO ESTESO

Un'istruzione che usa *l'indirizzamento esteso* contiene, nei suoi ultimi due byte, un indirizzo a 16 bit. Questo indirizzo può essere usato come puntatore della locazione di memoria per i dati richiesti, oppure può essere l'indirizzo al quale il programma dovrebbe saltare.
Un esempio del primo caso è

LD (1203H),A

il cui codice esadecimale è 32 03 12. Questa istruzione fa sì che nella locazione di memoria 1203 venga memorizzato il contenuto dell'accumulatore. Notate che, analogamente al caso del metodo di indirizzamento indiretto tramite registri, l'indirizzo è racchiuso fra parentesi. La notazione generale così è (nn), dove n è un byte a otto bit. Un'istruzione in cui nn rappresenta un indirizzo al quale il programma dovrebbe saltare è l'istruzione JP nn, per esempio

JP 1203H

il cui codice esadecimale è C3 03 12. Notate che in questa istruzione non trasferiamo dei dati, ma, piuttosto, il controllo di programma all'istruzione di indirizzo 1203. Quindi, questa volta nn non è scritto fra parentesi.

INDIRIZZAMENTO MODIFICATO IN PAGINA ZERO

Vi sono otto istruzioni dello Z-80 che utilizzano questo tipo d'indirizzamento. Queste istruzioni, dette istruzioni di RESTART (lett.: riparti), fanno sì che il controllo di programma venga trasferito a parti del programma chiamate subroutine. Di questo tipo di

trasferimento del controllo di programma parleremo più avanti. Tutte le istruzioni di Restart sono lunghe un byte. Il codice operativo specifica uno degli otto possibili indirizzi - 0000,0008,0010,0018,0020,0028,0030,0038 - uno per ogni istruzione di Restart. Dato che il byte d'indirizzo di ordine più elevato è sempre 00, il metodo d'indirizzamento è chiamato *"modificato in pagina zero"*. Lo scopo principale delle istruzioni di Restart è quello di accedere a subroutine che vengono usate spesso. I vantaggi di queste istruzioni consistono nel fatto che risparmiano tempo e spazio e possono essere forzate nel microprocessore durante un'interruzione. Le istruzioni analoghe di chiamata (Call) di subroutine usano tre byte invece dell'uno richiesto da un Restart. Ecco una semplice istruzione di Restart:

RST 10H

il cui codice esadecimale è D7. Questa istruzione trasferisce il controllo di programma alla subroutine situata a 0010 (decimale 16).

INDIRIZZAMENTO IMPLICITO

Alcune istruzioni dello Z-80 fanno riferimento automaticamente ad un particolare registro. Questo tipo di istruzione utilizza *l'indirizzamento implicito*. Il gruppo di istruzioni aritmetiche e logiche ad otto bit sono esempi di istruzioni a indirizzamento implicito, perchè implicano tutte operazioni sul contenuto dell'accumulatore.

L'istruzione ADD A,B

il cui codice esadecimale è 80, somma il contenuto del registro B all'accumulatore e carica il risultato nell'accumulatore stesso.

INDIRIZZAMENTO DI UN SINGOLO BIT

Il set di istruzioni dello Z-80 contiene molte istruzioni che indirizzano dei singoli bit all'interno di byte immagazzinati in memoria o nei registri. Queste istruzioni che manipolano i bit usano una combinazione di metodi d'indirizzamento. L'indirizzamento tra registri, indiretto tramite registri o indicizzato, specifica la locazione di memoria o il registro della CPU che contiene il byte cui ci si vuole riferire; un codice a tre bit interno al codice operativo dell'istruzione specifica poi il bit - bit 0,1,2,3,4,5,6 o 7 - dove i bit sono numerati all'interno del byte a partire da destra:

Byte	MSB						LSB
Numero del bit	7	6	5	4	3	2	1 0

Un esempio di istruzione d'indirizzamento relativo ai bit è

SET 3,B

il cui codice esadecimale è CB DB. Questa istruzione porta a livello logico 1 il bit numero 3 (decimale) del registro B. Molte delle nuove istruzioni dello Z-80, non implementate sul microprocessor 8080A della Intel, sono le istruzioni di BIT, SET e RESET, che utilizzano tutte il metodo d'indirizzamento di un singolo bit.

INDIRIZZAMENTO INDICIZZATO

Lo Z-80 ha due registri specializzati a 16 bit detti *registri indice*, che sono indicati come registri IX e IY. Essi vengono usati principalmente con il metodo di indirizzamento indicizzato. Tale indirizzamento è molto simile all'indirizzamento indiretto tramite registri,

per il fatto che il contenuto di un registro a 16 bit indica la locazione in memoria dei dati desiderati. L'unica differenza importante è che nel caso dell'indirizzamento indicizzato, si specifica anche uno spiazzamento di un byte nel primo byte dell'istruzione dopo il codice operativo. Questo byte è un numero complemento a due di otto bit che indica di quante posizioni (avanti o indietro rispetto al contenuto del registro indice) è "spiazzata", cioé spostata, la locazione di memoria cui si vuole accedere. Per esempio,

LD A,(IX+02H)

il cui codice esadecimale è DD 7E 02, carica l'accumulatore con il contenuto della locazione di memoria di due posizioni più alta rispetto alla locazione indirizzata da IX. L'istruzione

LD (IY+FFH),A

il cui codice esadecimale è FD 77 FF, memorizza il contenuto dell'accumulatore nella locazione di una posizione più bassa rispetto all'indirizzo nel registro IY, dato che FF è la rappresentazione complemento a due del valore decimale − 1. La Tabella 7-1 spiega più estesamente le istruzioni LD già indicate. L'indirizzamento indicizzato si indica con (IX+d) o (IY+d), dove d rappresenta il byte di spiazzamento espresso in complemento a due. Le parentesi indicano che IX+d e IY+d puntano una locazione di memoria.

La figura 7-1 spiega il significato di d nelle istruzioni che usano l'indirizzamento indicizzato, (per es. LD A,(IX+D)).

(IX+d)	Significato dello spiazzamento in relazione alla locazione di memoria (in decimale)
(IX+7FH)	127 byte più alta di IX
(IX+0FH)	15 byte più alta di IX
(IX+09H)	9 byte più alta di IX
(IX+01H)	1 byte più alta di IX
(IX+00H)	(IX)
(IX+FFH)	1 byte più basso di IX
(IX+FEH)	2 byte più bassi di IX
(IX+FDH)	3 byte più bassi di IX
(IX+FCH)	4 byte più bassi di IX
(IX+F0H)	16 byte più bassi di IX
(IX+E0H)	32 byte più bassi di IX
(IX+D0H)	48 byte più bassi di IX
(IX+80H)	128 byte più bassi di IX

Tabella 7-1

L'indirizzamento indicizzato è uno strumento molto potente per accedere a tabelle di dati in memoria. Tipicamente i registri IX ed IY sono caricati con l'indirizzo della prima

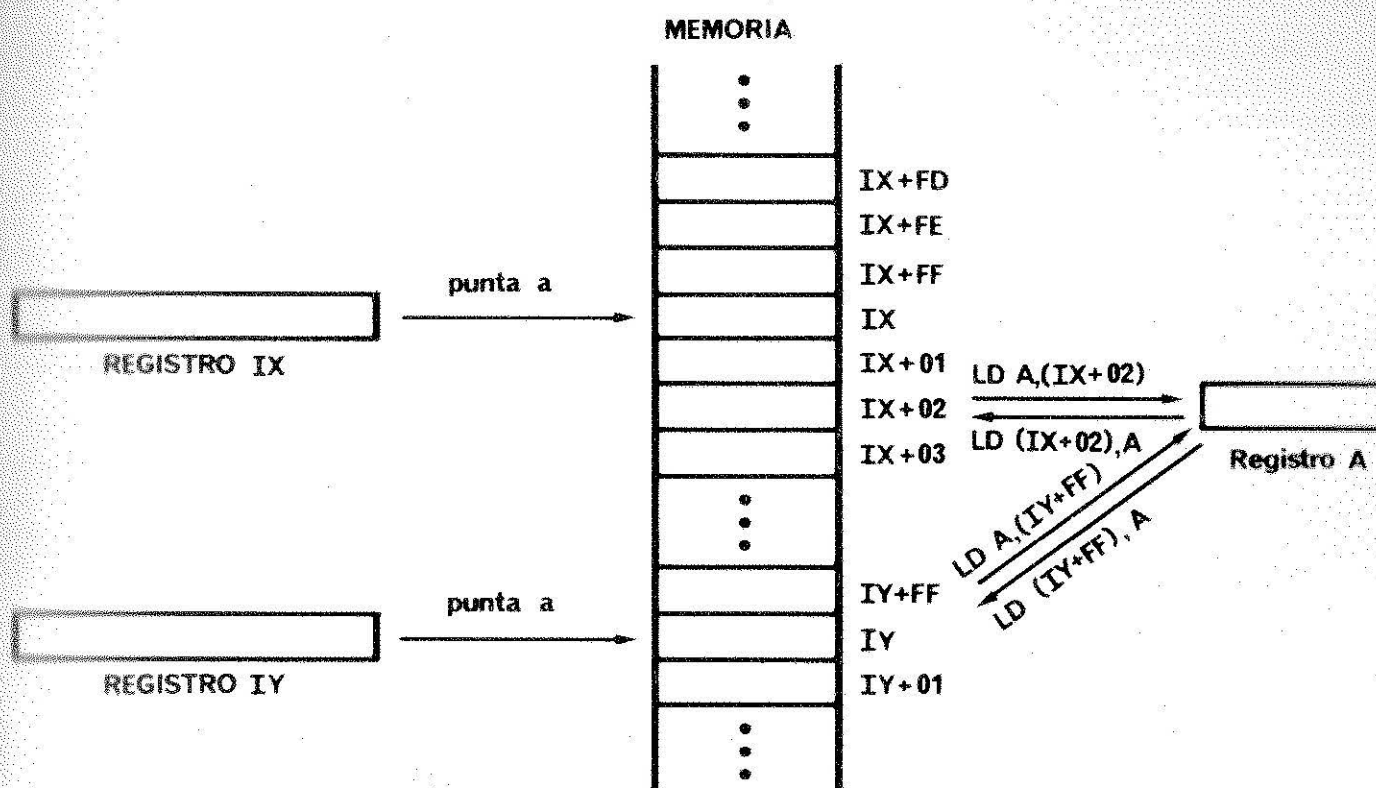


Figura 7-1

locazione della tabella, e poi tutte le altre locazioni della tabella sono ottenute facendo riferimento alla loro posizione relativa rispetto al punto d'inizio.
Cioè, il byte di spiazzamento cambia a seconda della locazione voluta, sempre riferendosi al punto d'inizio della tabella cui si vuole accedere. Questo chiarisce il fatto molto importante per cui l'esecuzione di una istruzione che utilizza l'indirizzamento indicizzato NON modifica i contenuti del registro indice.
Molti programmi che utilizzano l'indirizzamento indicizzato per accedere ad una tabella sono riportati alla fine di questo capitolo.

INDIRIZZAMENTO RELATIVO

L'indirizzamento relativo è un metodo di indirizzamento molto specializzato applicabile solo a quelle istruzioni di salto dette appunto di *salto relativo* (JR). Come nel caso dell'indirizzamento indicizzato il primo byte dopo il codice operativo è un numero ad 8 bit complemento a due che rappresenta uno spiazzamento rispetto un dato indirizzo. Si consideri l'istruzione

JR 09H

il cui codice esadecimale associato è: 18 09. Lo 09 è la distanza, cioé lo spiazzamento, tra la posizione della *istruzione che deve essere eseguita dopo il salto* e la posizone dell'istruzione che segue, in memoria, l'istruzione di salto. Cioè, questo è un salto incondizionato relativo ad una istruzione nove byte più avanti nel programma, rispetto all'istruzione che sarebbe stata normalmente seguita.
L'istruzione

JR FCH

il cui codice esadecimale è 18 FC, fa sì che il controllo di programma venga trasferito indietro di quattro byte contati a partire dall'istruzione che segue l'istruzione di salto, dato che FC è la rappresentazione complemento a due con otto bit di – 4. Nella Figura 7-2 vengono illustrate queste due istruzioni.

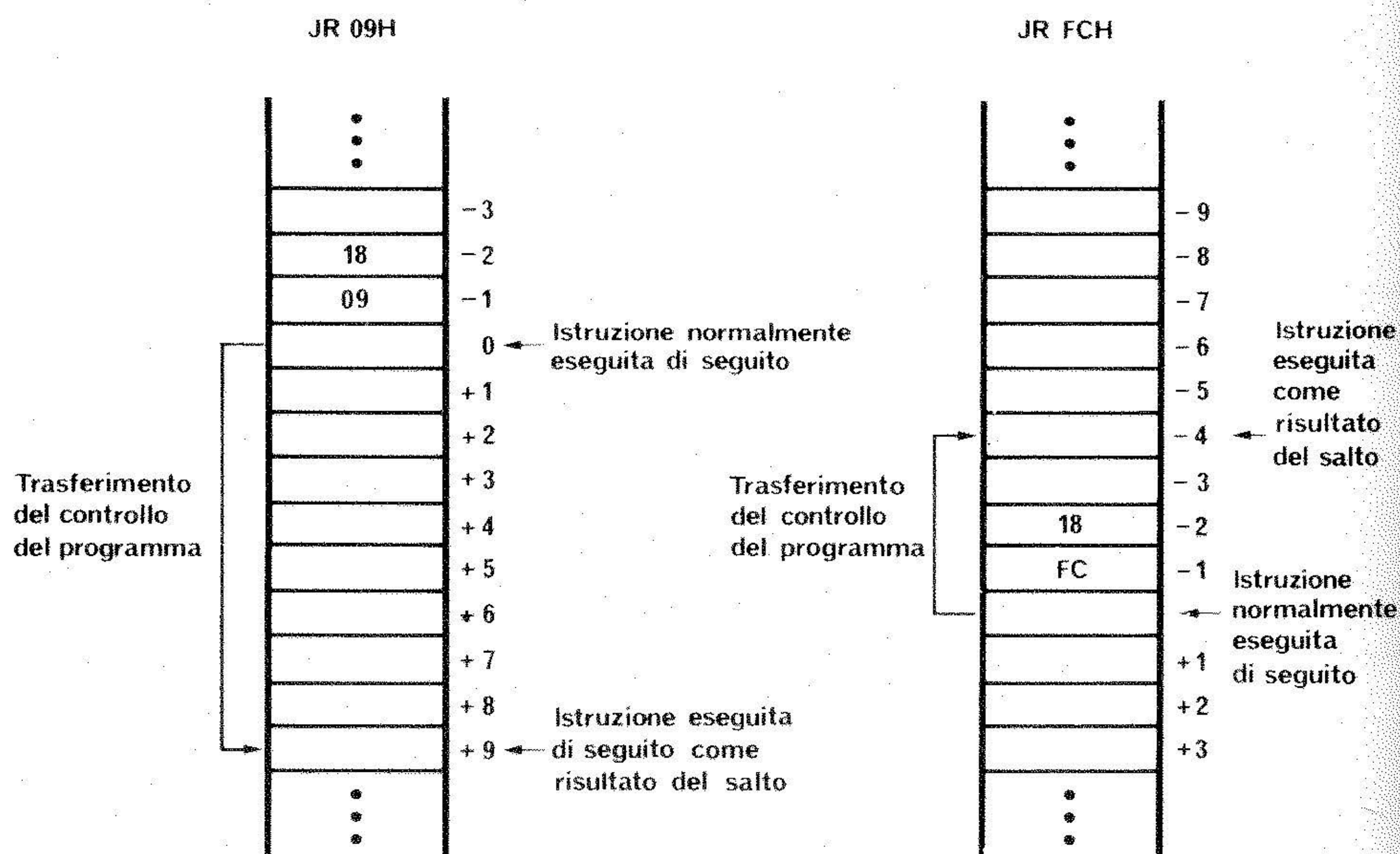


Figura 7-2

Il metodo d'indirizzamento relativo dello Z-80 è una caratteristica estremamente interessante in quanto permette di scrivere codici *rilocabili.* Si dice che un programma o un blocco di codici istruzioni è rilocabile se è scritto in modo indipendente dalla posizione in cui risiede fisicamente in memoria. Per controllare se un programma è rilocabile, basta spostare il programma, INVARIATO, in una nuova locazione di memoria. Se il programma viene eseguito senza problemi, il programma è *rilocabile.* Chiaramente, qualunque programma con una normale istruzione di salto, che usa l'indirizzamento esteso, non è rilocabile.
Un salto normale specifica un indirizzo assoluto, per cui lo spostamento del programma in una nuova locazione richiede che questo indirizzo assoluto venga cambiato prima dell'esecuzione. La procedura di cambiamento di tutti gli indirizzi assoluti, associata al cambiamento di posizione di un programma, è detta rilocazione del programma.
Un altro vantaggio dei salti relativi è che richiedono solo due byte di memoria, invece dei tre byte richiesti dai salti assoluti. I salti relativi sono però limitati fra +127 e −128 locazioni.
Con questo si conclude l'elenco dei metodi d'indirizzamento. Come avrete indubbiamente notato, sono state introdotte molte istruzioni nuove.
Abbiamo pensato che ogni metodo d'indirizzamento debba essere illustrato per lo meno con un esempio, anche se questo significa introdurre una nuova istruzione.
Vi assicuriamo che ritorneremo su ognuna di queste nuove istruzioni, trattandole in modo esauriente nei capitoli successivi.

TABELLE DEI GRUPPI DI ISTRUZIONI

Ora che siete al corrente di tutti i metodi d'indirizzamento dello Z-80, vorremmo farvi conoscere un metodo molto utile per visualizzare le istruzioni dello Z-80 ed i codici macchina esadecimali ad esse associati. Tali tabelle suddividono le istruzioni entro diversi gruppi, a seconda della loro funzione.
Esaminiamo prima la tabella del gruppo di caricamento a otto bit, Tabella 7-2.

		SORGENTE															
		IMPLICITO		TRA REGISTRI							INDIRETTO TRAMITE REG.			INDICIZZATO		ESTESO	IMME.
		I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	(nn)	n
DESTINAZIONE: TRA REGISTRI	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
	B			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
	C			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
	D			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
	E			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
	H			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
	L			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
INDIRETTO TRAMITE REGISTRO	(HL)			77	70	71	72	73	74	75							36 n
	(BC)			02													
	(DE)			12													
INDICIZZ.	(IX+d)			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
	(IY+d)			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
ESTESO	(nn)			32 n n													
IMPLICITO	I			ED 47													
	R			ED 4F													

Tabella 7-2. Gruppo di caricamento a 8 bit

Notate che sia la prima colonna a sinistra che la prima riga in alto contengono delle etichette (label) che indicano i diversi metodi d'indirizzamento. Sono due i metodi d'indirizzamento usati da ogni istruzione di caricamento a otto bit - uno per la destinazione (righe orizzontali) ed uno per la sorgente (colonne verticali). Supponiamo che vogliate spostare il contenuto del registro C nel registro D. Quindi D è la destinazione, e dovete cercare la riga contrassegnata D nella tabella. Cercate nelle colonne verticali finchè arrivate alla colonna del registro sorgente C: avete così trovato il codice macchina esadecimale corrispondente all'istruzione LD D,C che è 51. In ogni cella di questa tabella compare un codice esadecimale per il quale esiste un'istruzione di caricamento dello Z-80. Quindi, questa tabella vi dice QUALI istruzioni vengono implementate, nonchè i codici esadecimali ad esse relativi.

Vediamo alcuni esempi.

LD A,(IX+d) ha come codice esadecimale DD 7E d, dove il terzo byte d è il byte di spiazzamento utilizato per l'indirizzamento indicizzato

LD (nn),A ha come codice esadecimale 32 nn, dove la prima n è il byte LO e la seconda è il byte HI dell'indirizzo in cui deve essere memorizzato il contenuto di A.

LD (IY+d),n ha come codice esadecimale FD 36 d n, dove d è lo spiazzamento e n è il byte che verrà memorizzato nella locazione di memoria spiazzata dalla locazione indicata da IY.

LD (HL), (BC) non è implementata sullo Z-80.

Notate che nella tabella alcuni metodi d'indirizzamento non compaiono. Se in una tabella indicata a un certo gruppo di istruzioni un metodo d'indirizzamento non compare, vuol dire che quel gruppo non usa quel metodo d'indirizzamento. Perciò, possiamo vedere che vi sono quattro metodi d'indirizzamento non implementati dal gruppo di caricamento a otto bit: quello per bit singoli, quello relativo, quello immediato esteso e quello in pagina zero modificata.

IL GRUPPO DI CARICAMENTO A SEDICI BIT

Nella Tabella 7-3 vedete il gruppo di caricamento a 16 bit.

		SORGENTE									
		TRA REGISTRI							**IMM. EST.**	**ESTESO**	**INDIR. TRAM. REG.**
DESTINAZIONE		AF	BC	DE	HL	SP	IX	IY	nn	(nn)	(SP)
TRA REGISTRI	AF										**F1**
	BC								01 n n	ED 4B n n	**C1**
	DE								11 n n	ED 5B n n	**D1**
	HL								21 n n	2A n n	**E1**
	SP				**F9**		DD F9	FD F9	31 n n	ED 7B n n	
	IX								DD 21 n n	DD 2A n n	DD E1
	IY								FD 21 n n	FD 2A n n	FD E1
ESTESO	(nn)		ED 43 n n	ED 53 n n	**22** **n** **n**	ED 73 n n	DD 22 n n	FD 22 n n			
ISTRUZIONI PUSH → **INDIR. TRAM. REG.**	(SP)	**F5**	**C5**	**D5**	**E5**		DD E5	FD E5			

ISTRUZIONI POP (↑ colonna (SP))

Nota - Le istruzioni PUSH e POP modificano il registro SP dopo ogni esecuzione

Tabella 7-3. Gruppo di caricamento a 16 bit "LD" "PUSH" E "POP".

In questa tabella vi sono indubbiamente meno istruzioni che nella precedente, la Tabella 7-2. La maggior parte dei caricamenti a 16 bit implicano l'indirizzamento immediato esteso, o quello esteso, con pochissimi trasferimenti a 16 bit fra le coppie di registri. C'è solo una coppia di registri per cui è possibile effettuare l'indirizzamento indiretto tramite registri. Questo registro, STACK POINTER (SP), ha una funzione particolare di cui parleremo ora dettagliatamente.

Push e Pop dello Stack

Lo Stack In materia di computer, la parola STACK (pila o catasta) indica una particolare struttura di dati, che si può spiegare con un esempio preso dalla vita di tutti i giorni:

Siamo in un self-service. I vassoi puliti di cui i clienti si servono si trovano accatastati (in inglese "stack", appunto) sul banco. Il vassoio che conviene prendere è quello che sta in cima a questo mucchio. Perciò, man mano che vengono serviti nuovi clienti i vassoi usati sono stati lavati e asciugati, essi vengono "spinti" (pushed) o infilati sopra il mucchio (stack). La relazione che regola il comportamento della pila può essere così riassunta:

LAST IN, FIRST OUT: l'ultimo a entrare è il primo a uscire.

Questa regola (LIFO) è quella che governa gli stack come strutture di dati. Illustreremo questo nuovo concetto con un esempio, usando i byte della memoria del computer al posto dei vassoi. Di seguito è riportato un elenco di parte della memoria, con ogni locazione contrassegnata dal suo indirizzo. Notate che, parlando delle operazioni dello stack, gli indirizzi di memoria *aumentano* man mano che si scende verso il *basso*. Si tratta di una variante rispetto al nostro modo usuale di disegnare i diagrammi di memoria. Lo facciamo perchè lo stack pointer, registro SP, punta sempre al byte dello stack che ha l'indirizzo più basso.

SP	00
SP + 1	01
SP + 2	02
SP + 3	03
	⋮

Si può pensare a questo insieme di locazioni di memoria come ad uno stack in cui la locazione SP (che sta per Stack Pointer) rappresenta l'indirizzo del byte in cima allo stack. Vediamo che il contenuto della parte alta dello stack, (SP), è 00. Su questo stack di byte si possono eseguire due operazioni:

1. Estrarre i byte dalla parte alta (POP)
2. Inserire nuovi byte nello stack (PUSH)

Entrambe queste operazioni danno origine ad un nuovo byte in cima allo stack. Il microprocessore Z-80 ha due istruzioni, POP e PUSH, che eseguono le suddette operazioni per due byte alla volta.
Entrambe le istruzioni richiedono che una coppia di registri sia specificata come sorgente (per PUSH) o come destinazione (per POP) dei due byte di dati che vengono trasferiti. Gli esempi che seguono dovrebbero chiarirvi il tutto. La cosa più importante da ricordare è che il *byte in cima allo stack ha l'indirizzo più basso.*

ESEMPIO 1.

Ecco illustrata l'istruzione POP BC:

Prima dell'esecuzione

SP	00	B	?
SP+1	01	C	?
SP+2	02		
SP+3	03		

Dopo l'esecuzione

Non più nello stack	00	B	01
	01	C	00
SP	02		
SP+1	03		

L'esecuzione dell'istruzione POP BC ha i seguenti effetti:

1. Il byte in cima allo stack (SP) è memorizzato nel registro C
   C ←—— (SP)

2. Il secondo byte dello stack (SP+1) è memorizzato nel registro B
   B ←—— (SP+1)

3. Lo stack pointer (registro SP) viene aggiornato per indicare la nuova cima dello stack, eliminando così i due byte 00 e 01 dallo stack. Questo cambiamento di SP porta a sommare 2 allo stack pointer originale per arrivare a quello nuovo.
   SP ←—— SP + 2

Perciò le operazioni di POP fanno aumentare lo stack pointer! Notate che, anche se i byte 00 e 01 rimangono dov'erano prima dell'esecuzione di POP BC, la POSIZIONE dello stack in memoria è cambiata in modo tale da escluderli. Questo rappresenta una sottile differenza fra i byte e i vassoi che vengono estratti dal mucchio nel self-service, dato che i vassoi vengono fisicamente asportati dal mucchio.

ESEMPIO 2.

Vi illustriamo ora l'istruzione PUSH HL:

Prima dell'esecuzione

SP-2	?	Non sono parte		
SP-1	?	dello stack		
SP	00		H	03
SP+1	01	Stack	L	04
SP+2	02			

Dopo l'esecuzione

SP	04	Stack		
SP+1	03	nuovo		
SP+2	00		H	03
SP+3	01		L	04
SP+4	02			

Come potete vedere, PUSH HL ha l'effetto esattamente opposto dell'istruzione POP:

1. Il byte nel registro H è memorizzato nella locazione di memoria una posizione più in alto rispetto alla parte alta dello stack, SP-1
   (SP-1)←—— H

2. Il byte nel registro L è memorizzato nella locazione di memoria due posizioni più in alto rispetto alla parte alta dello stack, SP-2
   (SP-2)←—— L

3. Lo stack pointer viene aggiornato per indicare la nuova cima dello stack.
   SP←—— SP-2
   (Per ottenere il nuovo stack pointer, si sottrae due dallo stack pointer precedente).

Perciò: le operazioni di PUSH fanno *diminuire* lo stack pointer! Entrambi gli esempi illustrano dei particolari importanti da ricordare:

A. Lo stack va crescendo a partire dagli indirizzi alti verso quelli bassi. Cioè, "lo stack cresce verso il basso in memoria". POP fa aumentare e PUSH fa diminuire lo SP.

B. Vengono sempre inseriti ed estratti DUE byte. Tutte le operazioni di inserimento e di estrazione hanno luogo fra lo stack e le coppie di registri o i registri indice: AF,BC,DE,HL,IX o IY. Dallo stack vengono prelevati (pop) prima il byte LO e poi il byte HI della coppia di registri, mentre nello stack viene introdotto (push) prima il byte HI e poi il byte LO della coppia.

C. Le istruzioni di PUSH e POP differiscono da una normale istruzione di caricamento a sedici bit perchè il trasferimento dei dati è accompagnato da un aggiornamento del registro stack pointer.

D. Le istruzioni di PUSH e POP usano l'indirizzamento indiretto tramite registri perchè la locazione di memoria dei dati viene puntata dal contenuto del registro SP a sedici bit.

Lo stack e le operazioni ad esso associate, PUSH e POP, vengono usati per lo più insieme a trasferimenti di controllo dei programma detti SUBROUTINE CALL (chiamata di subroutine).
Questo argomento viene trattato nel capitolo che ha per oggetto i salti, le chiamate di subroutine e i rientri. Quindi ci soffermeremo ancora sullo stack in quel capitolo.
Presentandovi qui le operazioni dello stack, avevamo il fine di farvi capire bene il gruppo delle istruzioni di caricamento a 16 bit dello Z-80.

TRASFERIMENTO BLOCCHI E SCAMBI

Prima di concludere il capitolo, vorremmo mostrarvi altre due tabelle: quella relativa ai trasferimenti di blocchi e quella relativa agli scambi.

			SORGENTE INDIR. TRAM. REG. (HL)	
DESTINAZIONE	INDIRETTO TRAMITE REGISTRO	(DE)	ED A0	'LDI' – Load (DE)←(HL) Inc HL & DE, Dec BC
			ED B0	'LDIR,' – Load (DE)←(HL) Inc HL & DE, Dec BC, Repeat until BC = 0
			ED A8	'LDD' – Load (DE)←(HL) Dec HL & DE, Dec BC
			ED B8	'LDDR' – Load (DE)←(HL) Dec HL & DE, Dec BC, Repeat until BC = 0

Il Registro HL punta alla sorgente
Il Registro DE punta alla destinazione
Il Registro BC è il contatore di byte

Tabella 7-4. Gruppo di trasferimento a blocchi.

		INDIRIZZAMENTO IMPLICITO				
		AF'	BC', DE' & HL'	HL	IX	IY
IMPLICITO	AF	08				
	BC, DE & HL		D9			
	DE			**EB**		
INDIR. TRAM. REG.	(SP)			**E3**	DD E3	FD E3

Tabella 7-5. Istruzioni di scambio "EX" e "EXX".

Per quanto riguarda l'argomento del trasferimento di blocchi di dati, vi rimandiamo al Capitolo 6.
Le istruzioni di scambio danno luogo ad uno scambio di byte di dati fra registri a 16 bit o fra coppie di registri. Per esempio,

EX DE,HL

il cui codice esadecimale è EB, scambia il contenuto di DE con il contenuto di HL:

Prima dell'esecuzione di EX DE,HL

D 00
E 01
H 02
L 03

Dopo l'esecuzione di EX DE,HL

D 02
E 03
H 00
L 01

L'istruzione EX (SP),HL scambia il contenuto di HL con i due byte della parte alta dello stack; istruzioni analoghe vengono usate per i registri indice. Le istruzioni EXX e EX AF,AF' sono le uniche istruzioni dello Z-80 che coinvolgono il secondo set di registri non specializzati B',C',D',H',L',A' e F'. Potete quindi vedere che questi registri alternativi si possono usare solo per la memorizzazione temporanea dei registri principali, e non vi si può accedere con la stessa flessibilità con cui si può accedere al set principale.

INTRODUZIONE AGLI ESPERIMENTI E AGLI ESERCIZI

Alla fine di questo capitolo, abbiamo inserito sia esperimenti che esercizi per aiutarvi a rafforzare la vostra conoscenza della rappresentazione binaria complemento a due, dei metodi d'indirizzamento dello Z-80, delle operazioni relative allo stack, e dell'uso delle tabelle dei gruppi di istruzioni. Vi raccomandiamo di fare un po' di esercizi prima di eseguire gli esperimenti. Per questo motivo abbiamo messo gli esercizi di riepilogo prima degli esperimenti.

Gli esperimenti che eseguirete si possono così riassumere:

Esperimento N.	Commenti
1	Illustra come si può accedere a tabelle utilizzando l'indirizzamento indicizzato
2	Illustra i metodi alternativi per accedere a tabelle. Uno degli esempi è un programma che si modifica da solo.
3	Illustra le operazioni di PUSH e POP dello stack e le istruzioni di scambio.

ESERCIZI RIEPILOGATIVI

1. Calcolate il complemento (in rappresentazione complemento a due su otto bit) dei seguenti numeri binari a otto bit:

   a. 0 0 0 0 0 0 0 1
   b. 1 1 0 1 1 0 1 0
   c. 0 1 0 1 0 1 0 1
   d. 1 1 1 0 1 1 1 0
   e. 0 0 0 0 1 1 1 0
   f. 1 0 0 0 0 0 0 0
   g. 1 1 1 1 1 1 1 1

2. a. Qual è l'intero decimale più grande che può essere rappresentato con otto bit in complemento a due?
   b. Qual è l'intero decimale negativo più grande (in valore assoluto) che può essere rappresentato con otto bit in complemento a due?
   c. Rispondete alle domande a e b per un numero complemento a due a sedici bit.

3. Trovate il numero decimale corrispondente ai seguenti numeri in complemento a due su otto bit:

   a. 0 1 1 1 1 0 0 0
   b. 1 0 1 0 0 0 1 1
   c. 0 0 0 0 0 0 1 1
   d. 1 1 1 1 1 1 1 1
   e. 1 1 1 1 0 0 1 1
   f. 0 1 0 1 0 1 0 0
   g. 1 1 0 1 1 0 0 1

4. Trovate la rappresentazione complemento a due su otto bit dei seguenti numeri decimali:

a. 1
b. 16
c. −16
d. −128
e. 128
f. 121
g. −90

5. Vi presentiamo un elenco di istruzioni di salto relativo con i codici esadecimali ad esse corrispondenti. Usate queste informazioni per convertire tutte le istruzioni di salto "assoluto" (JP) in istruzioni di salto relativo (JR) nei programmi indicati di seguito.

Istruzione di salto relativo	Codice operativo esadecimale
JR	18
JR NZ	20
JR Z	28
JR PE	non implementata

a. Programma N. 9 nell'Esperimento N. 3 del Capitolo 6
b. Programma N. 10 nell'Esperimento N. 3 del Capitolo 6
c. Il programma N. 12 nell'Esperimento N. 5 del Capitolo 6

6. Per ognuna delle istruzioni seguenti, date i metodi d'indirizzamento usati con lo Z-80 ed i relativi codici esadecimali riferendovi alle Tabelle dei gruppi di istruzioni:

a. LD A,B
b. JR FBH (l'esercizio 5 dà il codice esadecimale)
c. LD A,(IX+06H)
d. LD (IX+06H),A
e. LD (1234H),A
f. LD (IX+09H),33
g. LD SP,HL
h. LD BC,0109H
i. LD (1030H),BC
j. LD IX,(1000H)
k. PUSH BC
l. POP IX

7. Indicate se le seguenti istruzioni sono implementate sul microprocessore Z-80. Se è così, indicate il relativo codice esadecimale.

a. LD AF,BC
b. LD B,(BC)
c. LD (BC),B
d. LD IX,IY
e. LD HL,BC
f. LD (1234H),56H
g. LD (1234H),B
h. LD (DE),45H
i. PUSH 1234H
j. POP SP

RISPOSTE

1. a. 1 1 1 1 1 1 1 1
   b. 0 0 1 0 0 1 1 0
   c. 1 0 1 0 1 0 1 1
   d. 0 0 0 1 0 0 1 0
   e. 1 1 1 1 0 0 1 0
   f. non esiste
   g. 0 0 0 0 0 0 0 1

2. a. 0 1 1 1 1 1 1 1 = + 127 (base 10)
   b. 1 0 0 0 0 0 0 0 = —128 (base 10)
   c. 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 = (2**15—1) (base 10) = il più alto
   d. 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 = (—2**15) (base 10) = il più basso

3. a. 120
   b. —93
   c. 3
   d. —1
   e. —13
   f. 84
   g. —39

4. a. 0 0 0 0 0 0 0 1
   b. 0 0 0 1 0 0 0 0
   c. 1 1 1 1 0 0 0 0
   d. 1 0 0 0 0 0 0 0
   e. non esiste
   f. 0 1 1 1 1 0 0 1
   g. 1 0 1 0 0 1 1 0

5. a.

Locazione di memoria	Codice oggetto		Codice sorgente
0120	0E 00		LD C,00H
0122	0D	LOOP:	DEC C
0123	20 FD		JR NZ.LOOP
0125	FF		RST 38 H

Per determinare quale indirizzo relativo si dovrebbe usare come <B2> per l'istruzione JR NZ, usate la seguente equazione:

indirizzo relativo = il complemento a due su otto bit dell'indirizzo assoluto dell'istruzione dopo l'istruzione di salto relativo meno l'indirizzo assoluto della destinazione del salto

= complemento a due su otto bit di (0125-0122)
= complemento a due su otto bit del byte 03
= 1 1 1 1 1 1 0 1
= FD

Notate che per poter definire correttamente un salto relativo tra le due locazioni, è necessario che la differenza tra i due indirizzi ad esse associati possa essere espressa in rappresentazione complemento a due su otto bit.
Questo programma è stato accorciato di un byte sostituendo un salto assoluto con un salto relativo.

5. b.

Indirizzo di memoria	Codice oggetto		Codice sorgente
0130	06 00		LD B,00H
0132	0E 00	LOOP 1:	LD C,00H
0134	0D	LOOP 2:	DEC C
0135	20 FD		JR NZ,LOOP2
0137	05		DEC B
0138	20 F8		JR NZ,LOOP1
013A	FF		RST 38H

Per l'istruzione JR NZ, LOOP 1: indirizzo relativo = complemento a due di 013A-0132
= complemento a due di 08
= F8

Per l'istruzione JR NZ, LOOP2: indirizzo relativo = complemento a due di (0137-0134)
= complemento a due di 03
FD

Questo programma è stato accorciato di due byte sostituendo due salti assoluti con due salti relativi.

5. c.

Locazione di memoria	Codice oggetto		Codice sorgente
0180	21 A0 01		LD HL,01A0H
0183	11 C0 01		LD DE,01C0H
0186	01 10 00		LD BC,0010H
0189	7E	LOOP:	LD A,(HL)
018A	B7		OR A
018B	28 05		JR Z,QUIT
018D	ED A0		LDI
018F	EA 89 01		JP PE,LOOP
0192	FF	QUIT:	RST 38H

Dei due salti di questo programma, solo, JP Z,QUIT può essere convertito in un salto relativo. Non esiste infatti tra le istruzioni di salto relativo, l'istruzione corrispondente a JP PE.
Quindi, per l'istruzione JP Z,QUIT:

indirizzo relativo = complemento a due dell'indirizzo della locazione dopo l'istruzione di salto
meno l'indirizzo della locazione di destinazione

=complemento a due di (018E-0193)

= (0193-018E)
= 05

Notate che, per saltare ad indirizzi più alti, l'elaborazione è più semplice perchè si può evitare l'operazione del complemento a due semplicemente cambiando l'ordine di sottrazione degli indirizzi.

6. a. Indirizzamento tra registri sia per la sorgente che per la destinazione - Codice esadecimale: 78
   b. Indirizzamento relativo - Codice esadecimale: 18 FB
   c. Destinazione: indirizzamento tra registri - Codice esadecimale: DD 7E 06
      Sorgente: Indirizzamento indicizzato - (IX+d)
   d. Destinazione: Indirizzamento indicizzato - (IX+d) - Codice esadecimale: DD 77 06
      Sorgente: indirizzamento tra registri
   e. Destinazione: indirizzamento esteso - (nn) - Codice esadecimale: 32 34 12
      Sorgente: indirizzamento tra registri
   f. Destinazione: indirizzamento indicizzato - (IX+d) - Codice esadecimale: DD 36 09 33
      Sorgente: indirizzamento immediato - n
   g. Destinazione: indirizzamento tra registri - Codice esadecimale F9
      Sorgente: indirizzamento tra registri
   h. Destinazione: indirizzamento tra registri - Codice esadecimale: 01 09 01
      Sorgente: immediato esteso - nn
   i. Destinazione: indirizzamento tra registri - Codice esadecimale ED 43 30 10
      Sorgente: indirizzamento tra registri
   j. Destinazione: indirizzamento tra registri - Codice esadecimale: DD 21 00 10
      Sorgente: immediato esteso
   k. La destinazione è (SP) e (SP-1): indirizzamento indiretto tramite registri
      Codice esadecimale: C5.
      La sorgente è la coppia di registri BC:
      indirizzamento tra registri.
   l. La destinazione è IX: indirizzamento tra registri - Codice esadecimale DD E1
      La sorgente è (SP) e (SP +1): indirizzamento indiretto tramite registri

7. Nessuna di queste istruzioni è implementata.

ESPERIMENTO N. 1

Scopo

Questo esperimento illustra la manipolazione di tabelle facendo uso dell'indirizzamento indicizzato.

Programma N. 15

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
0100	01 03 00		LD BC,0003H	;3 byte per riga
0103	FD 21 20 01		LD IY,0120H	;Indirizzo d'inizio della tabella=0120
0107	FD 7E 00	LOOP:	LD A,(IY)	;Memorizza la colonna 1 in A

```
010A    B7              OR A             ;E' zero?
011B    28 0A           JR Z,END         ;Se è zero, termina l'esecuzione
010D    FD 86 01        ADD A,(IY+01H)   ;Se no, somma la colonna 2
0110    FD 77 02        LD (IY+02H),A    ;Memorizza la somma nella colonna 3
0113    FD 09           ADD IY,BC        ;IY punta la riga successiva
0115    18 F0           JR LOOP          ;Ripeti la procedura precedente
0117    FF        END:  RST 38H          ;Rimanda il controllo al sistema operativo
```

Passo 1

Caricate il programma sopra riportato a partire dalla locazione 0100. Verificate di averlo caricato correttamente.

Passo 2

Questo programma manipola una tabella costituita da elementi (righe) lunghi ciascuno tre byte (colonne). Per ogni elemento (riga) della tabella viene effettuata la somma tra i primi due byte (prima due colonne) e il risultato è posto nel terzo byte (colonna tre). Questa procedura continua finchè si incontra una riga in cui il primo byte è 00. A quel punto, il controllo viene rimandato al sistema operativo del Nanocomputer. Per facilitarvi il tutto, supporremo per ora che gli addendi nelle colonne 1 e 2 siano piccoli abbastanza da non causare overflow.
Prendiamo ad esempio la tabella così costituita:

	Indirizzo	Col. 1	Col. 2	Col. 3
Riga 1	0120	01	02	?
Riga 2	0123	10	04	?
Riga 3	0126	23	13	?
Riga 4	0129	06	24	?
Riga 5	012C	00		

IY viene inizialmente caricato con 0120 e incrementato di 0003 ogni volta che si entra in una nuova riga.

Passo 3

Caricate le locazioni di memoria da 0120 a 012C con valori che compaiono nella tabella.

Passo 4

Eseguite il programma in modo passo passo guardando che cosa succede ai rgistri IY e A, e alle locazioni di memoria 0122, 0125, 0128 e 012B.

A questo punto avrete certamente notato come l'indirizzamento indicizzato ben si adatti alla manipolazione di tabelle di informazioni a due dimensioni. La riga della tabella viene stabilita tramite il contenuto del registro IY, mentre la colonna all'interno di ogni riga è specificata tramite lo spiazzamento da IY.

ESPERIMENTO N. 2

Scopo

Questo esperimento mostra che possono esserci molti modi alternativi di scrivere un programma che esegua una certa funzione. Una tecnica è quella dell'auto-modifica, secondo la quale un programma modifica da solo le sue istruzioni man mano che le esegue. Attenzione: *vi sconsigliamo di adottare questa tecnica, vi basti solo sapere che esiste.*

Programma N. 16

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
02 FD	01 07 00		LD BC,0007H	;BC = numero di colonne per riga
03 00	1E 06		LD E,06H	;Il registro E sarà usato come ;contatore per il numero di righe ;elaborate
03 02	FD 21 80 03		LD IY,0380H	;IY punta la riga in corso di ;elaborazione
03 06	21 11 03		LD HL,0311H	;HL punta la locazione del byte di ;spiazzamento dell'istruzione ;ADD A,(IY+d)
03 09	36 00	RIGA:	LD (HL),00H	;Inizializza lo spiazzamento
03 0B	3E 00		LD A,00H	;Inizializza il registro A
03 0D	16 06		LD D,06H	;Il registro D conta il numero di byte, ;di colonna sommati
03 0F	FD 86 d	COL:	ADD A,(IY+d)	;Qui usiamo d perché lo spiazzamento ;cambia man mano che viene eseguito ;il programma
03 12	00		NOP	;Nessuna operazione
03 13	34		INC (HL)	;Cambia lo spiazzamento
03 14	15		DEC D	;Aggiorna il contatore delle colonne
03 15	20 F8		JR NZ,COL	;Se non è zero, somma ancora
03 17	FD 77 06		LD (IY+06H),A	;Memorizza la somma nella colonna 7
03 1A	FD 09		ADD IY,BC	;Definisci IY alla riga seguente
03 1C	1D		DEC E	;Aggiorna il contatore della riga
03 1D	20 EA		JR NZ,RIGA	;Se non è zero, elabora la riga ;seguente
03 1F	FF		RST 38H	;Se è zero, restituisci il controllo al ;sistema operativo

Programma N. 17

Locazione di memoria	Codice oggetto		Codice sorgente
0320	01 07 00		LD BC,0007H
0323	1E 06		LD E,06H
0325	FD 21 80 03		LD IY,0380H
0329	3E 00	RIGA:	LD A,00H
032B	FD 86 00		ADD A,(IY)
032E	FD 86 01		ADD A,(IY+01H)
0331	FD 86 02		ADD A,(IY+02H)
0334	FD 86 03		ADD A,(IY+03H)

```
0337        FD 86 04              ADD A,(IY+04H)
033A        FD 86 05              ADD A,(IY+05H)
033D        FD 77 06              LD (IY+06H),A
0340        FD 09                 ADD IY,BC
0342        1D                    DEC E
0343        20 E4                 JR NZ,RIGA
0345        FF                    RST 38H
```

Programma N. 18

```
0360        1E 06                 LD E,06H
0362        FD 21 80 03           LD IY,0380H
0366        3E 00         RIGA:   LD A,00H
0368        16 06                 LD D,06H
036A        FD 06 00      COL:    ADD A,(IY)
036D        FD 23                 IN IY
036F        15                    DEC D
0370        20 F8                 JR NZ,COL
0372        1D                    DEC E
0373        FD 77 00              LD (IY+00H),A
0376        FD 23                 INC IY
0378        20 EC                 JR NZ,RIGA
037A        FF                    RST 38H
```

Passo 1

Guardate prima tutti i programmi che vi abbiamo mostrato e notate che eseguono tutti la stessa funzione. C'è una tabella memorizzata a partire dalla locazione di memoria 0386 con 6 righe orizzontali e 6 colonne verticali.
Ognuno di questi programmi elabora una per una tutte le righe sommando all'accumulatore i byte (colonne) di ciascuna di esse. Questi programmi variano per quanto riguarda l'occupazione di memoria e il tempo necessario per l'esecuzione, perché usano tecniche differenti. Esaminiamo da vicino ognuno di questi programmi.

Per tutti e tre vale la stessa struttura di programma generale rappresentata dal diagramma di flusso della Figura 7-3 riportato a pagina seguente.

I programmi differiscono l'uno dall'altro nelle soluzioni adottate per implementare i blocchi di elaborazione del diagramma di flusso, che sono stati contrassegnati con un asterisco.

Programma N. 16

L'algoritmo usato da questo programma modifica il byte di spiazzamento dell'istruzione ADD A,(IY+d). Più precisamente, nella coppia di registri HL viene memorizzata la locazione di memoria del terzo byte di questa istruzione, cioè dello spiazzamento. Inizialmente vengono posti a zero l'accumulatore e lo spiazzamento. Mentre un contatore di colonne controlla il numero di colonne elaborate, (IY+d) viene sommato all'accumulatore, e lo spiazzamento è incrementato, INC (HL). Quando tutte le colonne sono state sommate, il contatore delle colonne è decrementato a zero e l'istruzione JR NZ, quando questo si verifica, dà luogo ad un salto alle istruzioni che elaborano la riga seguente.

Come abbiamo già detto, questo programma si modifica da solo. Le quattro istruzioni del programma, attraverso cui viene effettuata questa operazione, sono:

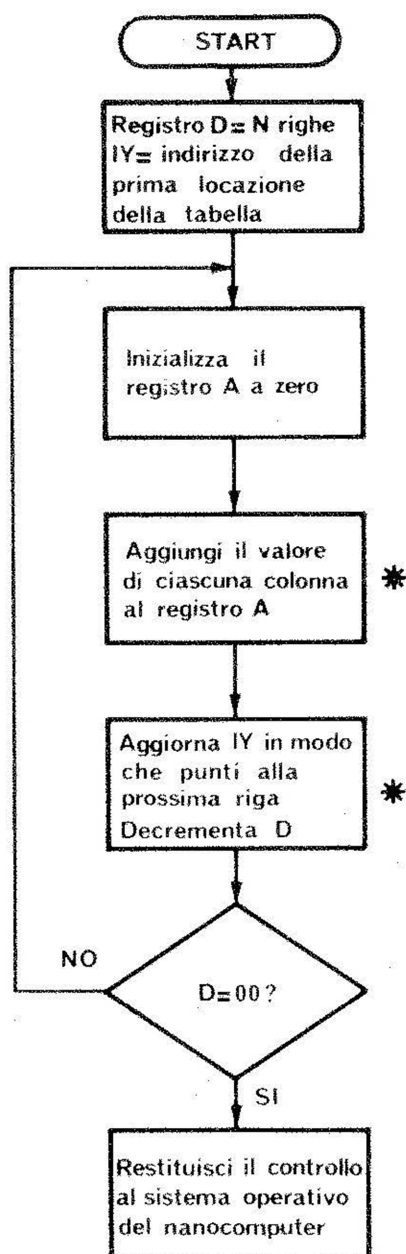


Figura 7-3

```
LD HL,0311H
LD (HL),00H
ADD A,(IY+d)     dove d è nella locazione 0311
INC (HL)
```

Due di queste istruzioni cambiano effettivamente il programma. Quindi il programma viene trattato alla stessa stregua dei suoi dati! Indubbiamente, questa è una tecnica che stimola l'immaginazione. I programmi possono scrivere nuovi programmi o alterarsi da soli, cambiando dinamicamente le loro caratteristiche. Comunque, vanno messi bene in evidenza i tre grossi svantaggi di questa tecnica:

a. I programmi che si modificano da soli sono spesso molto difficili da mettere a punto.

b. I programmi che si modificano da soli non possono risiedere in memorie a sola lettura.

c. I programmi che si modificano da soli sono spesso molto difficili da cambiare. Non è facile documentare tali programmi, per cui anche il programmatore che li ha ideati può incontrare grosse difficoltà nel ricordare i particolari del loro funzionamento.

Le tecniche di auto-modifica sono molto potenti ma devono essere usate con estrema attenzione.

Programma N. 17

L'algoritmo usato da questo programma è molto semplice e chiaro. Le sei istruzioni ADD A (IY+d) con d=00 ,..., 05 indicano molto chiaramente quello che il programma sta facendo. Sfortunatamente questa soluzione non è molto efficiente dal punto di vista dello spazio occupato, e, ovviamente, quando più aumenta il numero dei byte da sommare, tanto più questo metodo perde in efficienza.

Perciò, questo metodo è limitato alle applicazioni in cui si deve far riferimento a pochi byte, come è il caso, ad esempio, del Programma dell'Esperimento N. 1

Programma N. 18

Questo programma usa un algoritmo che non fa alcun uso di byte di spiazzamento diversi da zero. Il registro IY viene incrementato per ogni byte addizionato all'accumulatore. Va benissimo per questa applicazione e genera il programma più breve (rispetto al numero di byte). La tecnica qui usata nasconde un pò la presentazione dei dati sotto forma di tabella, perchè i dati vengono considerati come una matrice ad una sola dimensione (vettore) con IY come indice. Una delle ragioni per cui questa tecnica va bene in questo caso è che le locazioni a cui si deve far riferimento sono in sequenza, cioè esattamente una dopo l'altra in memoria.
Se le locazioni da sommare fossero IY+01,IY+09,IY+44,IY+56, questa tecnica dovrebbe chiaramente essere cambiata. Al contrario, la soluzione adottata nel Programma N. 17 andrebbe bene così com'è, dato che elenca semplicemente le colonne da sommare.
Nei paragrafi precedenti abbiamo fatto riferimento a numerose caratteristiche che possono essere attribuite a diversi programmi.Ora ve le riassumiamo:

a. SPAZIO - il numero di byte di memoria necessari per memorizzare un programma
b. TEMPO - il numero di stati della CPU necessari per eseguire un programma
c. FLESSIBILITA' - la facilità con cui un programma può essere cambiato
d. AUTO-MODIFICA - se il programma si modifica da solo o meno durante l'esecuzione.
e. SEMPLICITA' LOGICA - la facilità con cui il programma può essere letto e compreso.

Certamente questo non è un elenco esauriente, ma dovete prendere in considerazione ognuna di queste caratteristiche quando dovete definire il modo migliore per realizzare un programma. Di solito, è necessario accettare dei compromessi, perchè non si può ottimizzare una caratteristica senza sacrificarne un'altra. Per esempio, per scrivere programmi che risparmino sullo spazio in memoria, bisogna spesso accontentarsi di tempi di esecuzione più lenti. Esiste quindi uno compromesso *tempo-spazio*.
Analogamente, abbiamo visto che la semplicità logica spesso richiede più spazio, ecco quindi un altro compromesso *semplicità-spazio.*

Passo 2

Caricate il Programma N. 16 ed eseguitelo passo-passo, prestando particolare attenzione alla locazione di memoria 03 11, che contiene il byte di spiazzamento dell'istruzione ADD (IY+d). Usate la seguente tabella per i dati:

Memoria	Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6	Col. 7	
03 80	01	02	03	04	05	06	X	Riga 1
03 87	02	02	02	02	02	02	X	Riga 2
03 8E	01	03	01	03	01	03	X	Riga 3
03 95	03	03	03	03	03	03	X	Riga 4
03 9C	08	08	01	01	08	08	X	Riga 5
03 A3	04	04	04	04	08	08	X	Riga 6

Passo 3

Osservate attentamente il Programma N. 17 per vedere se è possibile renderlo più breve. Un miglioramento è quello di sostituire tutti i riferimenti a IY con dei riferimenti a HL. IN generale, per istruzioni analoghe, quelle che si riferiscono ad HL sono un byte più brevi di quelle per IY.

Speriamo che in questo esperimento vi siate accorti di come la programmazione sia molto più un'arte che una scienza. Un dato compito può essere eseguito da programmi diversi, alcuni più efficaci di altri per quanto riguarda il tempo o lo spazio o la semplicità logica o molte altre caratteristiche. L'arte consiste nel prendersi il fastidio di analizzare i pro e i contro di ogni alternativa, e nel trovare il compromesso migliore. Non è cosa insolita, per un programmatore, riscrivere un programma tre o quattro volte se il lavoro deve rispettare certe condizioni. Per esempio, una limitazione tipica è data dallo spazio di memoria. Una EPROM 2708 conterrà esattamente 1024 byte. Può darsi che un programma debba essere rifatto molte volte per utilizzare quei 1024 byte nel modo più efficace.

ESPERIMENTO N. 3

Scopo

Questo esperimento mostra l'uso delle operazioni di PUSH e POP dello stack, e delle istruzioni di scambio.

Programma N. 19

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
01 30		PUSH AF	;AF nella parte alta dello stack
		PUSH BC	;BC nella parte alta dello stack
		PUSH DE	;DE nella parte alta dello stack
		PUSH HL	;HL nella parte alta dello stack
		EX AF,AF'	;Scambia AF con AF'
		EXX	;Scambia le coppie di registri
		POP HL	;Parte alta dello stack in HL
		POP DE	;Parte alta dello stack in DE
		POP BC	;Parte alta dello stack in BC
		POP AF	;Parte alta dello stack in af
		RST 38H	;Restituisci il controllo del sistema operativo

Passo 1

Per quanto riguarda questo programma, vi invitiamo ad eseguire voi stessi l'assemblaggio manuale. Vi abbiamo perciò indicato un indirizzo d'inizio e le istruzioni espresse in codice

sorgente, cosicchè il codice oggetto è univocamente determinato. Per controllare il vostro assemblaggio manuale, fate riferimento a questa tabella dei contenuti della memoria:

0130	F5	C5	D5	E5	08	D9	E1	D1
0138	C1	F1	FF					

Passo 2

Caricate il codice oggetto partendo da 0130 e verificatene la correttezza.

Passo 3

Questo passo serve per posizionare lo stack nella memoria lettura/scrittura. Abbiamo due possibilità: possiamo lasciarlo dove già si trova (a 0F00 dopo aver premuto RESET) o possiamo cambiarlo. Queste sono le uniche istruzioni dello Z-80 che vi permettono di definire o modificare la locazione dello stack:

```
LD SP,HL
LD SP,IX
LD SP,IY
LD SP,nn
LD SP,(nn)
```

Potete inserirne una qualunque all'inizio del nostro programma per posizionare lo stack, la cui cima è appunto indicata dall'indirizzo contenuto nello stack pointer che tali istruzioni alterano.
Per la maggior parte dei programmi, terminata la procedura di inizializzazione, il registro SP viene aggiornato solo come risultato delle istruzioni PUSH e POP.
Utilizziamo una delle capacità che il sistema operativo del Nanocomputer possiede e carichiamo lo stack pointer con 0150, posizionando la luce di selezione a SP e memorizzando 0150.

Passo 4

Eseguite il programma in modo passo-passo con la luce di selezione nella posizione SP. Guardate lo stack, la cui lunghezza prima aumenta da 0 ad 8 byte e poi si riduce di nuovo:



Passo 5

Per riuscire a scoprire quali registri sono memorizzati e in quale ordine, inizializzate i registri come segue:

A=01	D=05
F=02	E=06
B=03	H=07
C=04	L=08

Eseguite il programma passo a passo finchè non è stata eseguita l'ultima istruzione di

PUSH, cioè finchè il PC=0134. Potete così verificare che le coppie di registri sono state inserite nello stack in questo modo:

01 48	L=08
	H=07
01 4A	E=06
	D=05
01 4C	C=04
	B=03
01 4E	F=02
	A=01
01 50	

Il byte HI di ogni coppia di registri è caricato nella locazione di memoria con l'indirizzo più alto.

Passo 6

La prossima istruzione da eseguire è

08 EX AF,AF' (AF'=A' e F')

Questa istruzione "scambia" il contenuto delle due coppie di registri AF e AF'. Prima di eseguire questa istruzione leggete e riportate qui sotto il contenuto di queste coppie di registri:

AF=0102 AF'= (noi abbiamo osservato 0000)

Ricordate che AF' può essere osservato usando il tasto ARS. Quando il led ARS è acceso, vengono visualizzati i registri alternativi.
Premete una volta sola il tasto SS per eseguire l'istruzione EX AF,AF'. Allora

AF= (noi abbiamo osservato 0000) ; AF'=0102

L'istruzione seguente è EXX, che scambia BC con BC', DE con DE', HL con HL'. Scrivete il contenuto di tutti questi registri, sia prima che dopo l'esecuzione di EXX:

			Le nostre osservazioni
Prima	BC=0304	BC'=	0000
	DE=0506	DE'=	0000
	HL=0708	HL'=	0000

Premete il tasto SS.

		Le nostre osservazioni	
Dopo	BC=	FFFF	BC'=0304
	DE=	FFFF	DE'=0506
	HL=	FFFF	HL'=0708

Passo 7

Le quattro istruzioni seguenti sono tutte istruzioni di POP, ciascuna delle quali carica i due byte della parte alta dello stack nelle coppie di registri specificate. Osservate la coppia di registri HL quando premete il tasto SS per eseguire POP HL. Il byte 08 nella locazione di memoria 0148 viene memorizzato nel registro L mentre 07, nella locazione di memoria 0149, viene memorizzato nel registro H.

Analogamente, le due istruzioni di POP che seguono memorizzano, nelle coppie di registri DE e BC, i valori 0304 e 0102, rispettivamente. Vediamo quindi che l'esecuzione di questo programma fa sì che nel set di registri alternativi vengano copiati i contenuti dei registri principali, lasciando questi ultimi invariati.

Passo 8

Vorremmo chiarire due particolari importanti riguardanti le operazioni dello stack:

1. PUSH E POP spostano SEMPRE 16 bit, cioè due byte di informazioni. Le istruzioni del tipo POP C e PUSH F non esistono.
2. L'ordine nel quale le coppie di registri vengono inserite sullo stack è opposto all'ordine nel quale esse devono essere estratte se si vuole che nei registri siano ripristinati i valori originari. Più precisamente

```
PUSH HL
PUSH DE
POP HL
POP DE
```

è una sequenza di istruzioni equivalente all'istruzione

```
EX DE,HL
```

Mentre,

```
PUSH HL
PUSH DE
.
.
.
Qualunque sequenza di istruzioni, anche istruzioni
che alterano DE e HL
.
.
.
POP DE
POP HL
```

conservano i registri DE e HL come erano prima che la sequenza delle istruzioni venisse eseguita. Questo tipo di operazione, anche se non sembra a prima vista, si dimostra invece molto utile, come vedrete più avanti studiando le subroutine.

CAPITOLO 8

SALTI, CHIAMATE E RITORNI

INTRODUZIONE

Le istruzioni di salto, di chiamata e di ritorno comprendono la classe di istruzioni dello Z-80 chiamate istruzioni di "branch". Tutte queste istruzioni fanno sì che il flusso delle istruzioni del programma venga trasferito in posizioni di memoria diverse da quelle che sarebbero state scelte se non si fosse verificata l'istruzione di salto. In questo capitolo, amplierete la vostra conoscenza di queste istruzioni al di là di quelle più semplici che avete già visto, cioè JP e JR. In particolare, imparerete le tecniche per usare le subroutine. Le istruzioni di salto, chiamata e ritorno sono riportate nella Tabella 8-1.

OBIETTIVI

Alla fine di questo capitolo, sarete in grado di:

- Dare la definizione di *trasferimento del controllo di programma*, evidenziando quanto succede al registro contatore di programma (PC);
- Dare la definizione di flag di *zero*, di *parità/overflow*, e di *segno*;
- Dare la definizione di *chiamata e ritorno dalla subroutine*, indicando quando succede al contatore di programma (PC), allo stack pointer (SP) e allo stack;
- Definire ed usare le *istruzioni di restart*.

TRASFERIMENTI DEL CONTROLLO DI PROGRAMMA

Voi avete ben chiaro in mente che un programma è semplicemente un insieme di byte di memoria che rappresentano istruzioni e dati. Normalmente queste istruzioni vengono eseguite in sequenza, cioè esattamente una dopo l'altra, finchè qualcosa (come un'istruzione di JP) cambia questo tipo di esecuzione. Vediamo da vicino come lo Z-80 esegue un programma memorizzato in qualche punto della memoria. La CPU dello Z-80 memorizza l'indirizzo dell'istruzione da eseguire successivamente nel registro contatore di programma (PC). Quindi, eseguire un programma significa ripetere i seguenti passi, finchè non viene eseguita l'istruzione di HALT:

PASSO 1. Trasferisci (PC) nel registro istruzioni interno allo Z-80. Ricordate che la notazione "(PC)" sta a significare "il contenuto della locazione di memoria indirizzata dal contatore di programma a 16 bit". (PC) è il primo byte istruzione, e perciò è un codice operativo che incomincerà a definire l'istruzione da eseguire. In alcuni casi, questo byte costituirà tutta l'istruzione.
In altri casi, la CPU dovrà leggere un altro byte prima di sapere quanti byte è lunga l'istruzione.

PASSO 2. Decodifica il primo byte dell'istruzione e decidi se è necessario leggere dei byte addizionali. Incrementa l'indirizzo in PC, in modo che esso punti alla locazione di memoria successiva, cioè al byte seguente del programma. Se la decodifica indica che l'istruzione è ad un solo byte, vai al PASSO 4.

PASSO 3. Continua a leggere (PC) e incrementa PC finchè non è stata letta tutta l'istruzione, (al massimo quattro byte).

PASSO 4. Esegui l'istruzione, poi ritorna al PASSO 1.

Notate che, prima di iniziare l'esecuzione di una istruzione, il PC contiene già l'indirizzo della locazione successiva. Come vedrete, tutte le istruzioni di branch, cioè i salti, le chiamate di subroutine, che indicheremo come CALL e i ritorni, agiscono direttamente sul registro PC per cambiare la sequenza nella quale le locazioni di memoria vengono lette ed eseguite come istruzioni.

ISTRUZIONI DI SALTO INCONDIZIONATO

Le istruzioni di salto fanno sì che il controllo di programma venga trasferito ad un indirizzo specificato dall'istruzione stessa. Dopo che il primo byte dell'istruzione di salto è stato letto e decodificato dalla CPU dello Z-80, viene determinato il tipo di istruzione. Quindi, l'esecuzione dell'istruzione procede con la lettura del byte o dei byte seguenti per determinare l'indirizzo di salto, dopo di che l'indirizzo di salto viene memorizzato nel registro PC. Quando la CPU inizia il ciclo di istruzione successivo, il PC contiene già l'indirizzo della istruzione che deve essere eseguita, indirizzo appunto ricavato dai byte di indirizzo contenuti nell'istruzione precedente (quella di salto). Vi mostriamo ora l'effetto che un'istruzione di JP ha sul PC:

Indirizzi	Istruzione	
1000	LD A	PC=1000 inizialmente
1001	00H	
→1002	INC A	PC=1002 dopo l'esecuzione di LD A,00H
1003	JP	PC=1003 dopo l'esecuzione di INC A
1004	02H	
1005	10	
		PC=1002 dopo l'esecuzione di JP 1002H (Con un'istruzione a tre byte non di salto, PC conterrebbe, dopo l'esecuzione di tale istruzione, il valore 1006).

Notate che l'istruzione JP coinvolge solo il registro PC. Non avviene nessun'altra operazione. Nel caso dell'istruzione JP, tutto l'indirizzo di salto è contenuto nell'istruzione come secondo e terzo (ultimo) byte. Perciò, eseguire il salto equivale a trasferire gli ultimi due byte istruzione nel registro PC. L'effetto dell'istruzione di salto sul controllo di programma è illustrato nella Figura 8-1.

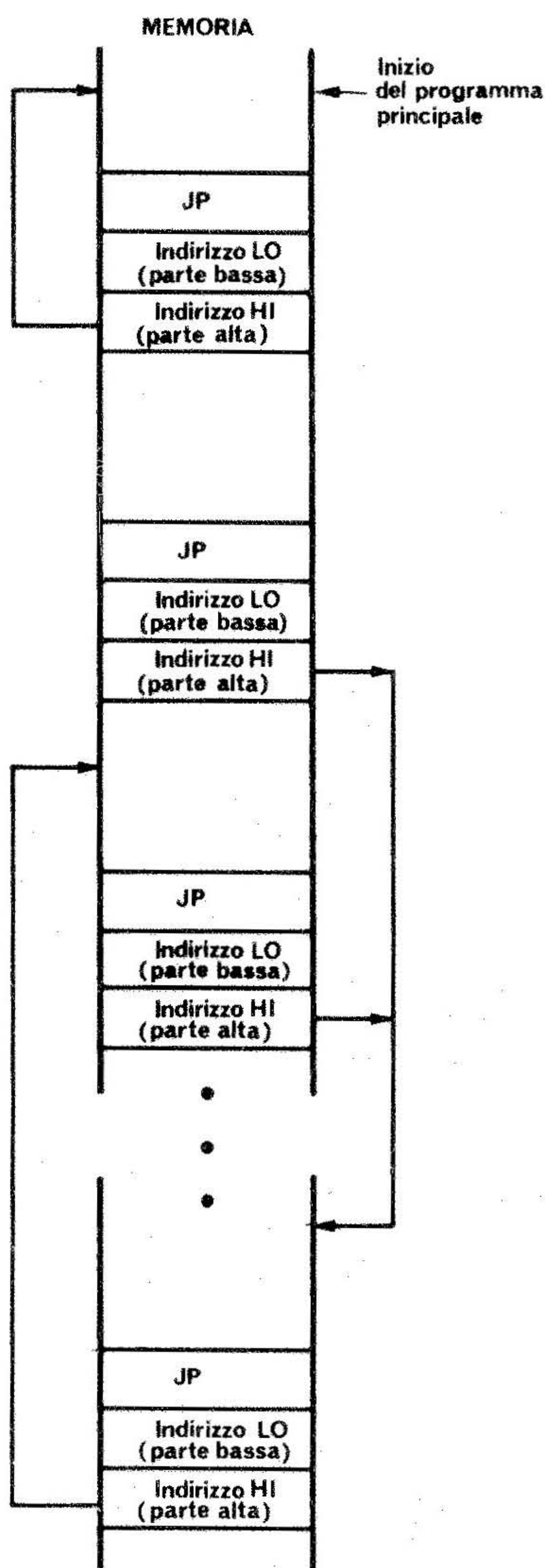


Figura 8-1

I salti possono avvenire "in avanti" o "all'indietro". In genere, più un programma salta, più sarà difficile da cambiare e da mettere a punto.

Vi sono due tipi di istruzioni di salto implementate sul microprocessor Z-80, quelle di SALTO ASSOLUTO e quelle di SALTO RELATIVO. L'istruzione di JP usata nell'esempio precedente è un salto assoluto perchè l'indirizzo di salto a due byte è completamente specificato all'interno dell'istruzione. I salti relativi (JR) specificano uno spiazzamento, in complemento a due, in un byte nell'istruzione. Quindi, l'esecuzione di un salto relativo implica determinare l'indirizzo di salto come somma dell'attuale valore del PC e del byte di spiazzamento. Vi mostriamo ora lo stesso programma di prima, ma in questo caso usiamo un salto relativo.

Indirizzi	Istruzione	
1000	LD A,	PC=1000, inizialmente
1001	00H	
►1002	INC A	PC=1002 dopo l'esecuzione di LD A,00H
1003	JR	PC=1003 dopo l'esecuzione di INC A
1004	FDH	
		PC=1002 dopo l'esecuzione di JR FDH (Un'istruzione a due byte, non di salto, avrebbe lasciato PC=1005 dopo l'esecuzione).

Notate che FD è il complemento a due di −3. Nell'istruzione di salto relativo, il byte di spiazzamento dà il numero dei byte prima (negativo) o dopo (positivo) la locazione puntata da un *contatore di programma normalmente aggiornato*. Nell'esempio di prima, il PC sarebbe diventato normalmente 1005 dopo l'esecuzione di un'istruzione a due byte non di salto. Tre byte *prima* di 1005 c'è la locazione 1002, la locazione a cui saltare, quindi lo spiazzamento è −3 o FD. Un errore che viene frequentemente commesso dai programmatori è quello di determinare il byte di spiazzamento relativo al PC prima dell'esecuzione dell'istruzione di JR. Questo è sbagliato e crea ovviamente dei problemi. *LO SPIAZZAMENTO PER UN'ISTRUZIONE DI SALTO RELATIVO VIENE SEMPRE DETERMINATO IN RELAZIONE ALLA PRIMA LOCAZIONE DOPO L'ISTRUZIONE DI JR A DUE BYTE.*
Vi elenchiamo qui alcune delle differenze esistenti fra i salti relativi e quelli assoluti:

1. L'istruzione di JP usa tre byte, un codice operativo ad un byte più un indirizzo a due byte. JR usa due byte per ogni istruzione, un codice operativo ad un byte più uno spiazzamento ad un byte.

2. L'istruzione di JP può fare in modo che il controllo di programma venga trasferito a qualunque locazione in memoria. L'istruzione di JR può fare in modo che il controllo di programma venga trasferito solo nelle locazioni di memoria comprese fra −128 e +127 byte dalla locazione di memoria immediatamente successiva al byte di spiazzamento. Questa limitazione è dovuta all'avere uno spiazzamento di un solo byte.

3. Le istruzioni di JP, generalmente non sono rilocabili, mentre le istruzioni JR lo sono. Una caratteristica molto importante dell'istruzione di JR è che essa fa riferimento a locazioni di memoria la cui posizione è calcolata in modo relativo all'istruzione JR stessa. Ciò implica che le istruzioni di JR mantengono la loro integrità a prescindere dalla loro locazione di memoria ASSOLUTA. Ogni programmatore che si sia trovato a spostare una grossa parte di un programma di un byte verso il basso in memoria, per inserire un'istruzione nuova, è in grado di apprezzare l'utilità di avere dei codici rilocabili (supponendo che l'assemblaggio venga fatto a mano).

Prima di prendere in considerazione i salti condizionati, vorremmo farvi notare che ci sono tre salti assoluti che usano l'indirizzamento indirètto tramite registri. Cioè, questi salti, JP (HL), JP (IX) e JP (IY), implicano che l'indirizzo di salto sia contenuto di una delle coppie di registri HL,IX o IY. Queste istruzioni, nonchè tutte le istruzioni di branch dello Z-80, sono contenute nella Tabella 8-1.

			CONDIZIONE									
			INCONDIZIONATO	RIPORTO	NON RIPORTO	ZERO	NON ZERO	PARITA' PARI	PARITA' DISPARI	SEGNO NEGATIVO	SEGNO POSITIVO	REG B ≠ 0
SALTO 'JP'	IMMEDIATO ESTESO	nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
SALTO 'JR'	RELATIVO	PC+e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
SALTO 'JP'	INDIRETTO TRAMITE REGISTRO	(HL)	E9									
SALTO 'JP'		(IX)	DD E9									
SALTO 'JP'		(IY)	FD E9									
CALL	IMMEDIATO ESTESO	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DECREMENTA B, SALTA SE NON ZERO 'DJNZ'	RELATIVO	PC+e										10 e-2
RITORNO 'RET'	INDIRETTO TRAMITE REGISTRO	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RITORNO DA INTERRUPT 'RETI'	INDIRETTO TRAMITE REGISTRO	(SP) (SP+1)	ED 4D									
RITORNO DA INTERRUPT NON MASCHERABILE 'RETN'	INDIRETTO TRAMITE REGISTRO	(SP) (SP+1)	ED 45									

Tabella 8-1. Gruppo Jump, Call e Return.

FLAG E SALTI CONDIZIONATI

Le istruzioni di JP e JR sono entrambe istruzioni di salto incondizionato. Questo significa che il controllo di programma viene SEMPRE trasferito all'indirizzo specificato, assoluto o relativo, quando viene eseguita l'istruzione.
I salti condizionati sono invece istruzioni che trasferiscono il controllo di programma ad una locazione diversa, a seconda che siano verificate o meno certe condizioni ben definite. Per esempio, l'istruzione JP NZ fa sì che si verifichi un salto se il flag di zero è a 0.
Quando vi abbiamo presentato questa istruzione nel Capitolo 6, non abbiamo parlato dei registri dei flag. Vorremmo farlo ora.

La CPU dello Z-80 contiene due registri di flag F e F', uno per ognuno dei due set di registri non specializzati. Ogni registro di flag contiene sei "FLAG", o bit di informazione, che vengono alterati o utilizzati individualmente dalle varie istruzioni dello Z-80. Diciamo che un flag è settato se il suo valore è 1, e resettato se è 0. Quattro dei sei flag vengono usati come condizioni per le istruzioni di salto, chiamata di subroutine e ritorno. Sono:

1. FLAG DI CARRY (RIPORTO) (C): questo flag viene alterato principalmente dalle istruzioni di addizione, sottrazione, rotazione e shift. Durante un'operazione di addizione, il flag è settato se c'è un riporto dal bit più significativo del risultato. Durante un'operazione di sottrazione, il flag è settato se c'è un riporto negativo dal bit più significativo del risultato. Molte istruzioni di rotazione e di shift considerano il flag di riporto come un nono bit da manipolare. Nei capitoli seguenti, tratteremo per esteso l'argomento delle istruzioni aritmetiche e logiche.
   Vi sono due istruzioni che manipolano direttamente il flag di carry:
   SCF - set del flag di carry, porta a 1 il flag C
   CCF - complementa il flag di carry. Se C=1, CCF resetta C (a 0). Se C=0, CCF setta C (a 1).

2. FLAG DI ZERO (Z): Lo stato di questo flag è influenzato da molte istruzioni. Le operazioni che cambiano l'accumulatore di solito influenzano il flag di zero, settandolo se l'accumulatore è zero, e resettandolo se l'accumulatore non è zero. L'istruzione BIT setta il flag di zero se un bit specificato è zero, e lo resetta se il bit è uno.
   L'istruzione di confronto, CP, controlla l'uguaglianza di un certo byte con l'accumulatore. Se c'è uguaglianza, il flag di zero è settato; altrimenti è resettato. Di altre istruzioni che coinvolgono il flag di zero, parleremo più avanti. Il flag di zero è l'unico flag che crea un pò di incertezze e confusione per quanto riguarda le condizioni in base a cui il flag è zero o uno.
   La ragione di questo è il modo in cui il flag viene manipolato: flag di zero uguale a uno significa che il risultato era zero e flag di zero uguale a zero significa che il risultato non era zero. Perciò, l'istruzione JP NZ significa salta se il RISULTATO NON E' ZERO; o, per quanto riguarda il flag di zero, salta se il FLAG DI ZERO E' ZERO. Flag di zero uguale a zero significa che il risultato di un'operazione precedente non era zero. Questa cosa crea molta confusione. Il consiglio migliore è suggerirvi di imparare a memoria quanto vi abbiamo detto e di stare molto attenti quando usate il flag di zero come condizione per il salto. L'importante è ricordare che la condizione NZ si riferisce al RISULTATO dell'operazione precedente, e NON al valore del flag.

3. FLAG DI SEGNO (S): questo flag è semplicemente una copia del bit più significativo del risultato di un'operazione, risultato che in genere, ma non sempre, è memorizzato nell'accumulatore.
   Lo scopo del flag S consiste nell'indicare se il risultato considerato come un numero in complemento a due è positivo o negativo.
   Il flag S è settato nel caso negativo, resettato nel caso positivo.

4. FLAG DI PARITA'/OVERFLOW (P/V): il flag P/V viene utilizzato per due scopi:
   a) per indicare la parità del risultato di una istruzione logica, di rotazione, di shift o di input.
   b) per indicare se si è verificato overflow al termine di una operazione aritmetica in complemento a due.

La parola *parità* si riferisce al numero di bit, in un byte,che sono ad uno. Se il numero di 1 è dispari (odd), allora si dice che quel byte ha *parità dispari.* Se il numero di 1 è pari (even), si dice che quel byte ha *parità pari.* Ad esempio la parità di FF è pari, mentre la parità di 01 è dispari.

Il flag di parità è settato se la parità del risultato è pari (even), ed è resettato se la parità del risultato è dispari (odd).
Nel Capitolo 6, abbiamo parlato di come individuare il verificarsi di un overflow con riferimento ad operazioni aritmetiche eseguite in logica complemento a due. Se la somma di due numeri positivi complemento a due determina come risultato un numero negativo, è settato il flag di overflow. Analogamente, il flag è settato se la somma di due numeri negativi determina un risultato positivo.
Il flag V è resettato se non vi è overflow.
E' importante comprendere la differenza tra il flag di carry (C) e quello di overflow (V). Se due numeri binari sono sommati o sottratti, il flag C è usato per individuare un overflow.

Se due numeri in complemento a due sono sommati o sottratti, è usato il flag V per individuare un overflow. Questi due flag *non* sono intercambiabili.
Ecco un esempio di verifica di quanto detto:

	1 1 1 1 1 0 1 1	rappresentazione complemento a due di −5
	+1 1 1 1 0 0 0 0	rappresentazione complemento a due di −16
Somma	1 1 1 0 1 0 1 1	rappresentazione complemento a due di −21

C=1 perchè c'è un riporto, inteso come superamento della capacità di contenimento di un registro od una locazione di memoria ad 8 bit.
V=0 perchè −21 è un risultato corretto, e non c'è overflow.

5. FLAG DI HALF-CARRY (H) E FLAG DI SOTTRAZIONE (N): i due flag che ancora ci restano sono flag specializzati, usati solo per l'aritmetica basata su di uno schema di codifica binaria detto sistema decimale codificato binario (BCD). Per ora vi diremo solo che i due flag si chiamano flag di half-carry (H) e flag di sottrazione (N). Entrambi sono importanti per l'istruzione di aggiustamento decimale dell'accumulatore (DAA). Nè il flag H nè il flag N vengono usati insieme ad operazioni di salto, come accade per i primi quattro flag, cioè ad essi non fa riferimento alcuna delle istruzioni di salto condizionato.
In questo diagramma vi diamo il formato del registro dei flag:

S	Z	X	H	X	P/V	N	C

dove X indica che il bit non ha un valore significativo. Talvolta X viene indicato come "condizione senza importanza" perchè non vi viene mai prestata attenzione. La "Z80-CPU Programming Reference Card" (libretto di riferimento delle istruzioni della CPU-Z80) della SGS-ATES, che avrete trovata allegata a questo libro, riassume in una tabella tutte le istruzioni dello Z-80 che alterano i flag. E' un ottimo riferimento, che vale la pena di imparare ad usare.

Abbiamo già parlato dell'istruzione di salto condizionato JP NZ, che trasferisce il controllo di programma ad un indirizzo, definito all'interno dell'istruzione, se il flag di zero (Z) è a 0. Per ognuno dei flag, S,Z,P/V, e C c'è un'istruzione di salto condizionato JP, la cui operazione è legata allo stato del flag corrispondente:

JP NZ - - - Salto se il flag Z è resettato (risultato non zero)

JP Z - - - Salto se il flag Z è settato (risultato zero)

JP NC - - - Salto se il flag C è resettato (nessun riporto)

JP C - - - Salto se il flag C è settato (riporto)

JP PO - - - Salto se il flag P è resettato (parità dispari o assenza di overflow)

JP PE - - - Salto se il flag P è settato (parità pari o presenza di overflow)

JP P - - - Salto se il flag S è resettato (risultato positivo)

JP M - - - Salto se il flag S è settato (risultato negativo)

Vi sono quattro istruzioni di salto relativo condizionato - JR NC,JR C, JR NZ e JR Z - oltre all'istruzione di salto relativo incondizionato, JR. Vediamo le due istruzioni di salto relativo, le cui azioni sono legate allo stato del flag di carry (C). L'effetto di queste istruzioni JR NC e JR C, è indicato in figura 8-2.

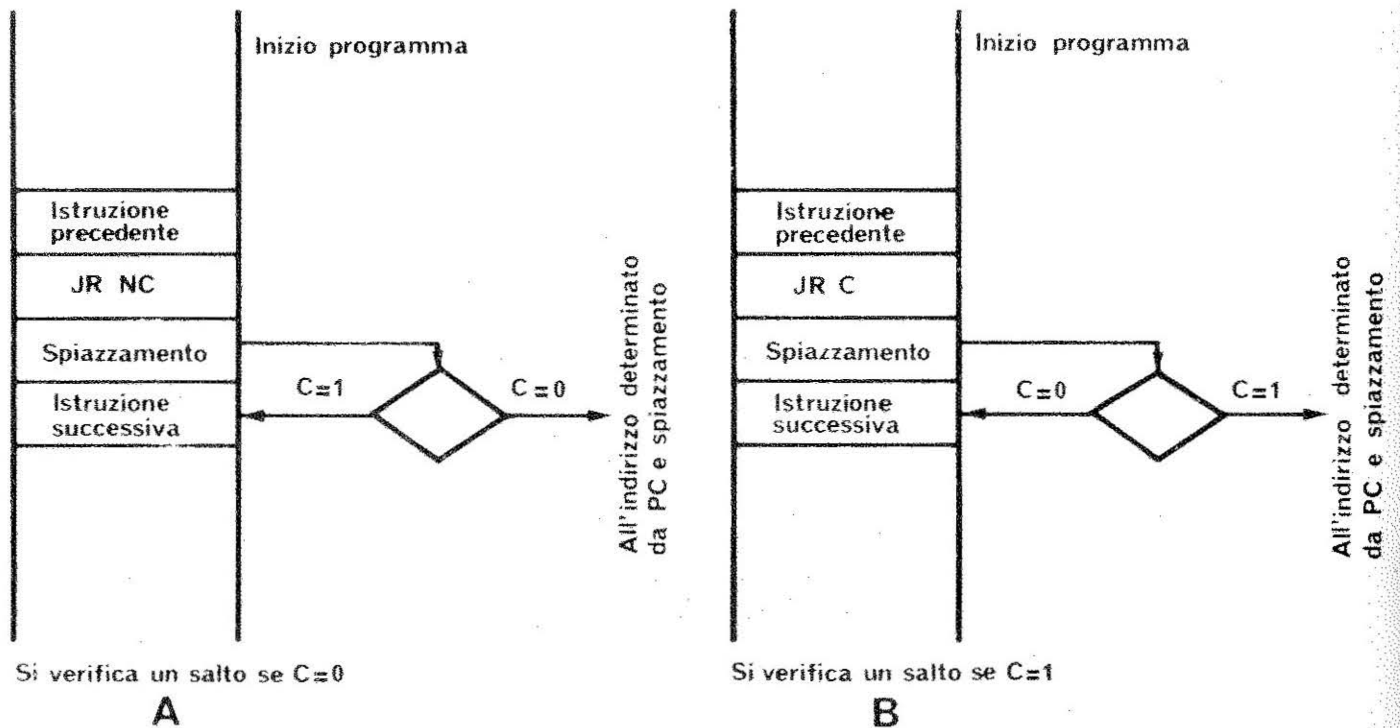
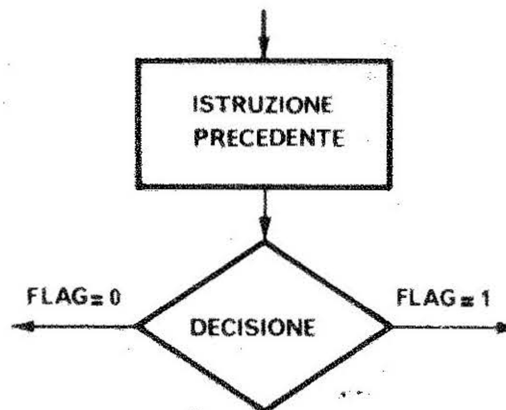


Figura 8-2

Il simbolo di decisione, riportato nella figura 8-2, che viene spesso usato nei diagrammi di flusso dei programmi, indica che quello che succede in seguito dipende dallo stato di una condizione (in questo caso un flag). Sia in questo capitolo che in quelli successivi, vedrete molti esempi di programmi che usano i salti condizionati.



Esiste anche un'istruzione particolare di salto relativo condizionato, il cui codice mnemonico è DJNZ. L'istruzione viene eseguita in tre passi:

a. Il registro B viene decrementato (di 1)
b. Viene fatto un controllo per vedere se il registro B è 00
c. Se B non è 00, avviene il salto. Altrimenti, viene eseguita l'istruzione successiva.

Il diagramma di flusso della figura 8-3 mostra come viene eseguita l'istruzione DJNZ.

Quando si programma, può capitare spesso di aver bisogno di eseguire lo stesso gruppo di istruzioni per un certo numero di volte. L'istruzione DJNZ serve a questo scopo. La Figura 8-4 vi mostra come si può usare l'istruzione DJNZ.

Come vedrete, questa istruzione è molto utile.

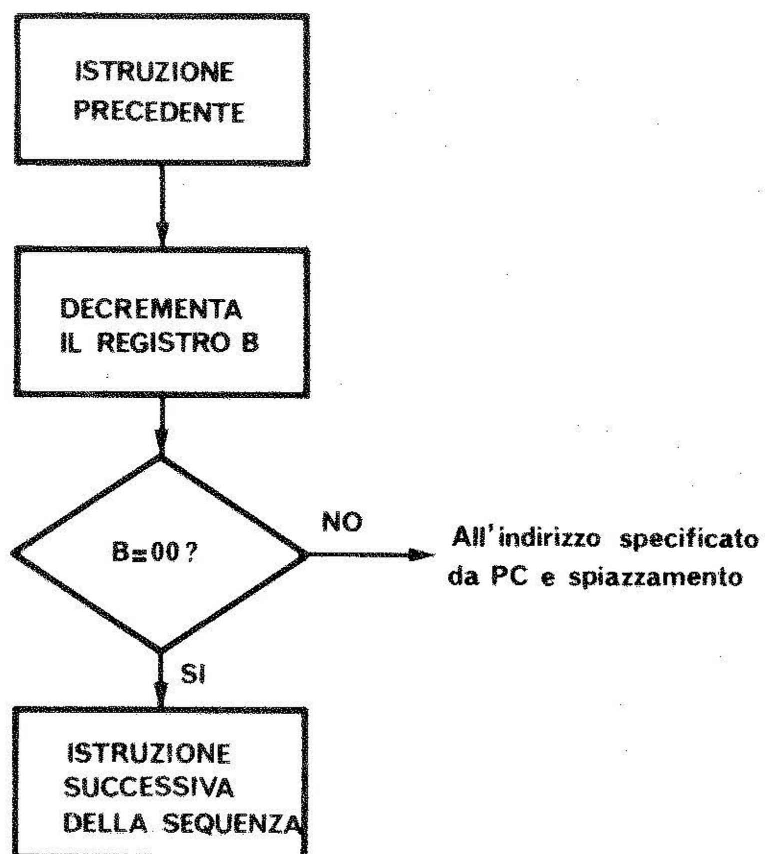


Figura 8-3

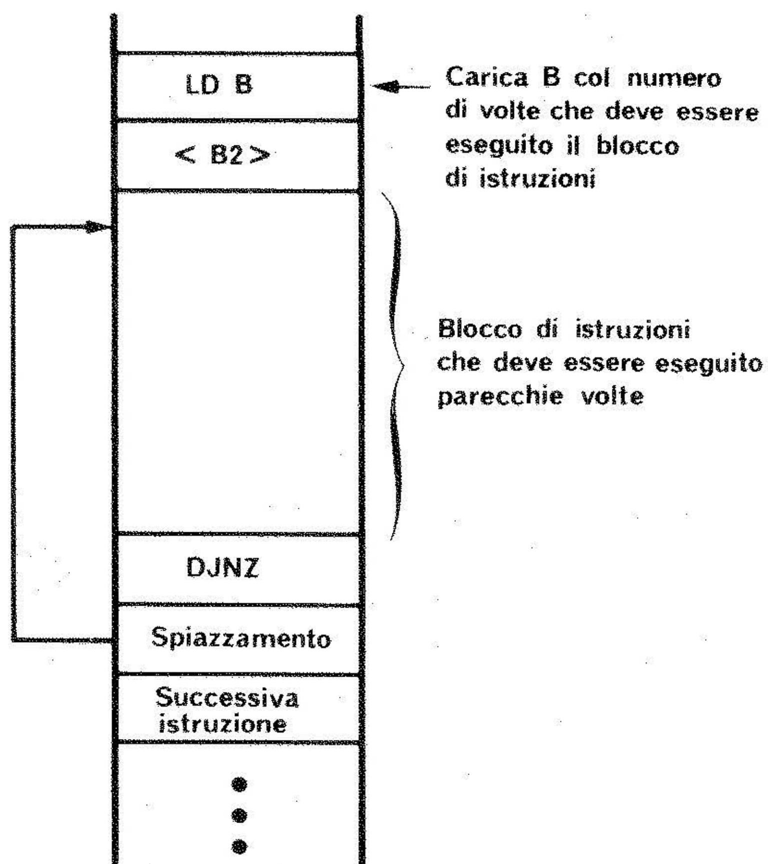


Figura 8-4

CHIAMATE E RITORNI

Un'istruzione di chiamata (Call) è un'istruzione di salto che "si ricorda da dove è saltata". L'istruzione di ritorno (RET) fa sì che la CPU riprenda l'esecuzione del programma dalla istruzione che segue l'ultima istruzione di chiamata eseguita. Osserviamo la Figura 8-5.

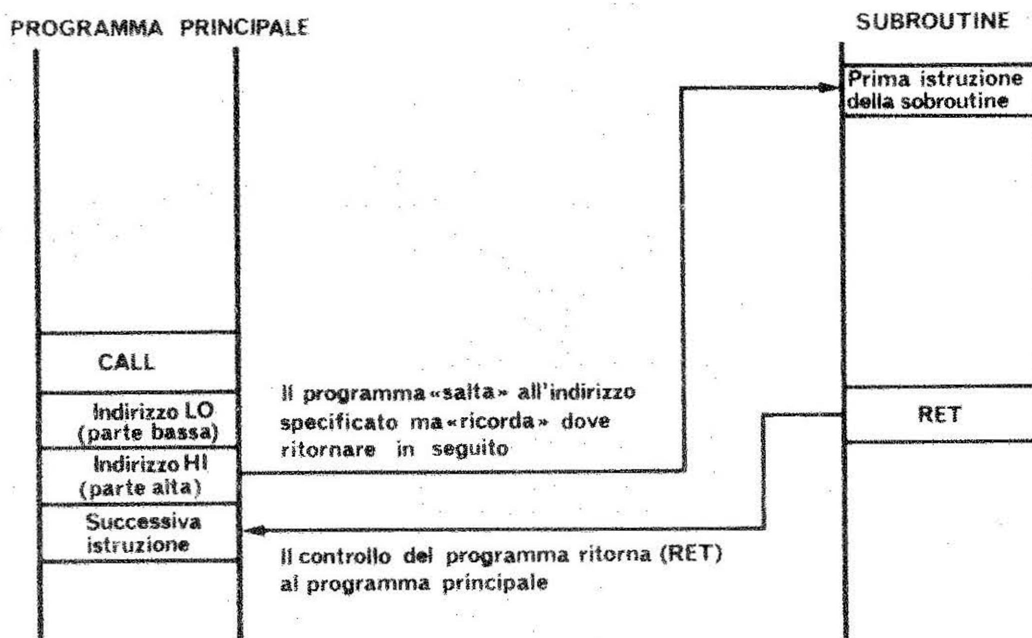


Figura 8-5

Come immaginerete, le istruzioni di CALL e RET sono molto utili perché permettono allo stesso blocco di codice (subroutine, cioé sottoprogramma) di essere eseguito, grazie all'uso di istruzioni di call, da molti punti diversi di un programma. I loop di ritardo vengono spesso scritti come subroutine.
Una subroutine di ritardo può essere scritta per generare un certo ritardo, mettiamo un millisecondo. Ogni volta che il programma richiede un ritardo di n millisecondi, la subroutine viene richiamata n volte.
I produttori di componenti, coloro che sviluppano il software, nonchè molti utenti di computer raccolgono le subroutine utili perché possano essere utilizzate da diversi programmatori. Queste raccolte si chiamano Biblioteche. Tali biblioteche potrebbero contenere ad esempio subroutine di moltiplicazione e divisione, subroutine per elaborare valori trigonometrici, logaritmici e funzioni esponenziali, ed altre subroutine per compiti specializzati.
Ora che sapete che cosa fanno le istruzioni di CALL e RET, dovete imparare come lo fanno. Vediamo prima l'istruzione di CALL. Proprio come l'istruzione JP addr, l'istruzione di CALL specifica l'indirizzo di una locazione di memoria nel secondo e nel terzo byte dell'istruzione. Questi due byte, quando viene eseguito il CALL, vengono caricati nel registro PC per effettuare il trasferimento del controllo di programma. Comunque, prima che PC venga aggiornato (prima cioé che venga effettuato il salto), l'istruzione di CALL esegue un primo passo che la distingue dall'istruzione di JP. Per ricordarsi a che punto bisogna ritornare nel programma principale, PC viene introdotto, salvato nello stack - prima il byte HI, poi il byte LO - *mentre il PC punta ancora l'istruzione che segue l'istruzione di CALL.*

La subroutine viene eseguita finchè non si incontra un'istruzione di ritorno. Tale istruzione di ritorno incondizionato, RET, fa sì che nel PC vengano caricati i due byte della parte alta dello stack - quello più in alto diventa il byte d'indirizzo LO, e quello che

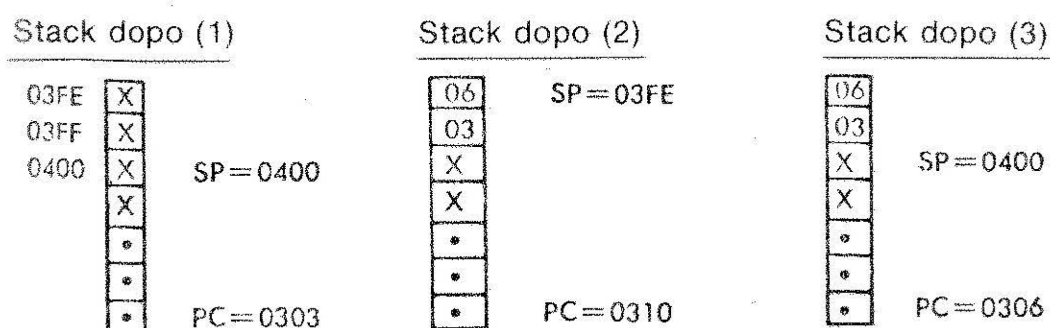
segue diventa il byte d'indirizzo HI. Se lo stack è stato usato correttamente, l'istruzione di RET dovrebbe trovare in cima allo stack i byte inseriti sullo stack dalla precedente istruzione di chiamata. Quindi il controllo di programma ritorna alla prima istruzione dopo CALL. Per illustrarvi il tutto, facciamo l'esempio di un breve programma che faccia uso delle istruzioni di CALL e di RET.

Indirizzo			
03 00	LD SP	Inizio del programma: inizializza lo stack	SP=04 00 (1)
03 01	00 H	pointer a 04 00	PC=03 03
03 02	04		
03 03	CALL	Chiama la subroutine a 0310: inserisci 03 06	SP=03 FE (2)
03 04	10 H	in cima allo stack; decrementa SP di 2; carica	PC=03 10
03 05	03	in PC il valore 03 10	
03 06	HALT	Alt	
03 10	INC A	Inizio della subroutine	SP=03 FE
03 11	INC B		PC=03 11
03 12	INC C		
03 13	DEC D		
03 14	DEC E		PC= 03 14 (3)
03 15	RET	Ritorno al programma principale: estrai i due byte in cima allo stack inserendoli in PC; incrementa SP di 2	SP= 04 00 PC= 03 06

STACK: lo stack è modificato solo da queste istruzioni:

(1) 03 00 LD SP
(2) 03 03 CALL
(3) 03 15 RET

(Notate che, nei diagrammi seguenti, gli indirizzi *aumentano* scendendo verso il basso).



(X indica un dato sconosciuto e che non ha importanza)

Tre trabocchetti possono trarre in inganno anche i programmatori più esperti:

1. Lo stack aumenta in modo tale che si espande verso il basso nella memoria finchè incomincia a "mangiarsi" il programma che sta usando, o i dati su cui il programma sta operando; oppure lo stack aumenta in modo tale che tenta di espandersi nella ROM o nella PROM e non può più accettare nuovi byte.

2. La subroutine chiamata inizia a manipolare lo stack in modo tale che l'istruzione di RET viene eseguita con una coppia sbagliata di byte nella parte alta dello stack. Chi sa dove manderà allora la CPU il controllo del programma?

3. Vi dimenticate di inizializzare lo stack pointer e quindi lo stack si trova in qualche punto casuale della memoria. In particolare, "in qualche punto" potrebbe essere nel mezzo del vostro *programma* o in un punto dove non esiste memoria RAM.

Tutti e tre questi casi di uso non corretto dello stack possono dar luogo al fatto che vengono interpretati dei dati come dei codici operativi con conseguenze del tutto imprevedibili ma sicuramente spiacevoli.

Anche con questi rischi, la possibilità di poter chiamare le subroutine è estremamente utile ed è un'eccellente pratica da sviluppare.

I programmi strutturati intorno alle subroutine di solito sono più facili da capire, più facili da modificare, e più facili da mettere a punto di quelli che non fanno uso di subroutine. Nell'esperimento N. 2 vi verrà spiegato quando è il caso di usare le subroutine. Analogamente all'istruzione di JP, anche per le istruzioni di CALL e RET esistono delle corrispnondenti istruzioni condizionate,la cui esecuzione è legata allo stato di uno dei flag. Le potete trovare nella Tabella 8-1.
Nella Tabella 8-2 sono elencate le istruzioni di restart (riparti), RST N. Si tratta di istruzioni ad un byte che eseguono una chiamata ad un indirizzo specifico interno al codice operativo. In altre parole, le istruzioni RST sono chiamate incondizionate ad un byte. L'indirizzo a due byte necessario in un'istruzione di CALL è sostituito dall'indirizzamento in pagina zero modificata. A seconda dei tre bit interni al codice operativo, si può specificare uno tra otto indirizzi esadecimali di subroutine:

0000,0008,0010,0018,0020,0028,0030,0038.

INDIRIZZO DI CHIAMATA		CODICE OPERATIVO	
	0000_H	C7	'RST 0H'
	0008_H	CF	'RST 8H'
	0010_H	D7	'RST 10H'
	0018_H	DF	'RST 18H'
	0020_H	E7	'RST 20H'
	0028_H	EF	'RST 28H'
	0030_H	F7	'RST 30H'
	0038_H	FF	'RST 38H'

Tabella 8-2. Gruppo Restart

INTRODUZIONE AGLI ESPERIMENTI

Questo insieme di esercizi è dedicato alle tecniche di programmazione che utilizzano le varie istruzioni di branch dello Z-80. I primi esperimenti presentano le istruzioni, dandovi dei semplici programmi da eseguire, durante la cui esecuzione osserverete il comportamento del registro contatore di programma (PC), degli altri registri e delle altre locazioni di memoria utilizzate. Gli ultimi esperimenti trattano alcuni problemi specifici, come quello del passaggio dei parametri alle subroutine o quello dell'uso di tabelle di salto. Gli esperimenti che eseguirete possono essere così riassunti:

Esperimento N.	Commenti
1	Illustra l'uso dell'istruzione di DJNZ in una routine di ritardo. E' anche illustrato come vengono usati i breakpoint.
2	Mostra come convertire il programma dell'esperimento N. 1 in una subroutine, e presenta un tipico programma di chiamata. Viene trattato il problema di quando usare le subroutine.
3	Illustra l'istruzione di RST N.
4	Illustra quattro tecniche per il passaggio dei parametri alle subroutine.
5	Illustra la tecnica dell'uso di tabelle di salto.

ESPERIMENTO N. 1

Scopo

Questo esperimento illustra l'istruzione di DJNZ, usandola per implementare un loop di ritardo.
Inoltre, viene illustrata la tecnica per inserire, usare e togliere i punti d'interruzione (breakpoint) per fermare l'esecuzione del programma.

Programma N. 20

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
01 00	06 09		LD B,09H	;Inizializza il registro B
01 02	0E FF	LOOP 1:	LD C,FFH	;Inizializza il registro C
01 04	16 FF	LOOP 2:	LD D,FFH	;Inizializza il registro D
01 06	15	LOOP 3:	DEC D	;Il loop più interno
01 07	20 FD			;decrementa il registro D
01 09	0D		JR NZ,LOOP 3	;Il loop intermedio decrementa
			DEC C	;il registro C
01 0A	20 F8		JR NZ,LOOP2	
			DJNZ LOOP 1	;Il loop più esterno decrementa
01 OC	10 F4			;il registro B
01 0E	FF		RST 38H	;Restituisci il controllo al ;sistema operativo del ;Nanocomputer

Passo 1

Caricate il programma ed eseguitelo, dando al registro B diversi valori iniziali. Come potete vedere, noi abbiamo usato 09. Notate che potete ampiamente variare il tempo in cui il display resta spento cambiando il valore iniziale del registro B. Nell'Appendice II vi viene chiaramente spiegato come calcolare il tempo di esecuzione di un dato programma. Noi abbiamo calcolato i seguenti tempi di esecuzione dando al registro B tre valori iniziali diversi.

Registro B	Tempo totale di esecuzione del programma nell'Esperimento N. 1
01	0,442 secondi (approssimativamente)
09	3,661 secondi (approssimativamente)
FF	105,47 secondi (approssimativamente)

L'istruzione DJNZ decrementa automaticamente il registro B. Viene quindi salvata un'istruzione, cioé DEC B, usando questo salto specializzato.

Passo 2

Supponete di voler osservare che cosa succede al registro B mentre il programma esegue l'istruzione di DJNZ. Potreste avanzare passo-passo, decrementando C e D, con C che viene decrementato 255 volte per ogni volta che viene decrementato D.
Questa operazione vi farebbe perdere però un bel po' di tempo: l'alternativa migliore è inserire un punto d'interruzione nel programma subito prima che venga eseguita l'istruzione di DJNZ. Per esempio, l'interruzione potrebbe essere inserita alla locazione 010C, dato che l'istruzione posta dove è inserito il punto d'interruzione non viene eseguita prima dell'arresto del programma. Dopo aver opportunamente caricato PC, premete GO in modo da far partire il programma. Il display si spegne per un breve intervallo di tempo, quindi quando viene incontrato il punto d'interruzione, il programma si ferma e il display ritorna attivo. A questo punto mettendo la luce di selezione sulla posizione BC e premendo il tasto SS, potrete vedere il registro B decrementarsi. Premete di nuovo GO e l'esecuzione si arresterà la prossima volta che il PC sarà uguale all'indirizzo del punto d'interruzione, cioè quando l'istruzione di DJNZ sta per essere eseguita di nuovo. Descriviamo più in dettaglio tale operazione.

Per inserire il punto d'interruzione, entrate nel Breakpoint Mode premendo il tasto BRK. Il led BRK dovrebbe accendersi e dovreste vedere uno zero sul display dei dati. Premete e tenete premuto il tasto INC. Dovreste vedere lo 0 cambiare in 1,2,3, fino a 7, e poi ritornare a 0. Se non sono stati settati altri punti d'interruzione, non dovrebbero apparire altre cifre nè sul display dei dati nè su quello degli indirizzi. Incrementate il contatore del punto d'interruzione (l'unico digit visualizzato) fino a leggere 3. Inserite 010C e premete LA. Dovreste allora vedere:

010C 310

010C è l'indirizzo del punto d'interruzione, 10 è il contenuto di quell'indirizzo (il codice operativo di DJNZ), e 3 è il valore del contatore del punto d'interruzione. Questo punto d'interruzione avrebbe potuto essere inserito in una qualsiasi posizione da 0 a 7, del contatore. Noi abbiamo arbitrariamente scelto 3.
Uscite dal Breakpoint Mode premendo di nuovo il tasto BRK. Il led di BRK si spegnerà. Eseguite il programma alla massima velocità iniziando alla locazione 0100. Il display dovrebbe spegnersi momentaneamente, e poi accendersi per mostrare:

010C 10

Cioé, PC=010C, e l'istruzione da eseguire successivamente è DJNZ. Posizionate la luce di selezione su BC e premete una volta il tasto SS. Il registro B dovrebbe diminuire di 1 e

Il valore del registro PC dovrebbe essere 0102. Per vedere l'istruzione di DJNZ eseguita una seconda volta, premete ancora GO. L'esecuzione si arresterà ancora a PC=010C. In questo modo vi è possibile avanzare passo passo nel programma interrompendo la sua esecuzione ogni volta che viene letta l'istruzione di DJNZ.

Passo 3

Eliminate il punto d'interruzione: mettetevi in Breakpoint Mode, cercate il punto d'interruzione che va cancellato usando il tasto INC, e infine premete GO.

ESPERIMENTO N. 2

Scopo

Questo esperimento mostra come convertire il programma dell'esperimento N. 1 in una subroutine, e introduce un esempio di programma che chiami la subroutine. Viene trattato anche il problema di quando è opportuno usare le subroutine.

Programma N. 21

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
01 1B	31 FF 01	LD SP,01FFH	;Definisci l'area di stack
01 1E	06 03	LD B,03H	;Specifica il valore del registro B
01 20	CD 02 01	CALL DELAY	;Chiama la subroutine
01 23	3E 00	LD A,00H	;Carica nell'accumulatore 00
01 25	FF	RST 38H	;Restituisci il controllo al sistema ;operativo

Subroutine delay

Locazione di memoria	Codice oggetto		Codice sorgente
0102	0E FF	DELAY:	LD C,FFH
0104	16 FF	LOOP2:	LD D,FFH
0106	15	LOOP3:	DEC D
0107	20 FD		JR NZ,LOOP3
0109	0D		DEC C
010A	20 F8		JR NZ,LOOP2
010C	10 F4		DJNZ DELAY
010E	C9		RET

Passo 1

Il programma dell'esperimento N. 1 appare qui con la dicitura SUBROUTINE DELAY. Lo abbiamo convertito in una subroutine sostituendo, fra l'altro, l'istruzione finale RST 38H con l'istruzione RET.
Quindi ora la subroutine restituisce il controllo al programma che ha effettuato la chiamata indicato in questo esperimento come PROGRAMMA, invece di ritornare al

sistema operativo del Nanocomputer. Sono state fatte altre due modifiche. Per prima cosa la label LOOP1 è stata cambiata in DELAY. Questo è stato fatto per ragioni puramente estetiche, dato che sembrava più "auto-documentante" per il programma principale, chiamare una subroutine DELAY invece che la subroutine LOOP1. In effetti questa modifica non era realmente necessaria. La seconda modifica è consentita nel togliere l'istruzione LD B dalla subroutine. Abbiamo spostato questa nel programma di chiamata, in modo che quest'ultimo potesse specificare la lunghezza del ritardo caricando il registro B prima di trasferire il controllo alla subroutine DELAY. Questa modifica non era richiesta per trasformare il programma in una subroutine, comunque, permette di utilizzare in modo molto più flessibile la nostra subroutine.
In particolare, la subroutine può essere usata per generare 256 valori diversi di ritardo, a seconda del contenuto del registro B al momento della chiamata. Questa tecnica, per cui il programma di chiamata definisce dei valori che devono essere utilizzati da una subroutine, è detta PARAMETER PASSING (passaggio dei parametri). In questo caso, il parametro è il contenuto del registro B, che determina la durata del ritardo. Vi sono altri modi per passare i parametri alle subroutine, modi che vengono trattati nell'esperimento N. 4.
In conclusione, riassumiamo le modifiche necessarie per trasformare un programma in una subroutine ed identifichiamo anche gli elementi necessari che un programma di chiamata deve avere.

Per convertire un programma in una subroutine:

L'unica modifica necessaria è inserire una istruzione RET (condizionata o incondizionata) dove si vuole terminare la subroutine e ritornare al programma di chiamata.
Notate che se viene a sostituire delle istruzioni di salto JP (3 byte) o JR (2 byte), l'istruzione RET (1 byte) può comportare un certo risparmio per quanto riguarda l'occupazione della memoria.
Nel peggiore dei casi, il programma viene allungato di un byte per ogni nuova istruzione di RET.

Per un programma di chiamata:

E' assolutamente necessario specificare esplicitamente in che punto della memoria deve venire a risiedere lo stack del sistema. E' possibile effettuare questa operazione attraverso l'istruzione a tre byte

LD SP,<B3> <B2>

Se il programma non usa in altro modo lo stack, cioè se non ci sono istruzioni di PUSH e POP nel programma, le chiamate delle subroutine costano tre byte di memoria per definire lo stack. Se il programma usa lo stack per altri scopi, lo stack dovrebbe essere già comunque definito.
Per ogni chiamata di una subroutine, sono normalmente necessari tre byte. (Le istruzioni di RST si possono usare solo per casi speciali).
Se l'istruzione di salto a subroutine sostituisce una istruzione di salto assoluto (JP), il programma non viene allungato. Nel peggiore dei casi, il programma si allunga di tre byte per ogni salto a subroutine.

Caricate il *Programma* e la *Subroutine* agli indirizzi indicati. (La subroutine dovrebbe essere stata già caricata nell'esperimento N. 1).

Passo 2

Inserite un punto d'interruzione alla locazione 010E, cioé subito prima dell'istruzione di RET. Eseguite il programma passo-passo, partendo dalla locazione 011B. Guardate il registro SP subito prima e subito dopo l'esecuzione dell'istruzione di CALL DELAY.

Prima: SP= ____________
Dopo: SP= ____________

Noi abbiamo osservato che, prima del CALL, SP=01FF e dopo il CALL, SP=01FD.

Passo 3

Osservate i due byte della parte alta dello stack, cioé le locazioni 01FE e 01FD. Dovreste vedere che l'indirizzo dell'istruzione che segue l'istruzione di Call Delay è stato inserito nello stack cioé (SP) = (01FD) = 23 e (SP+1) = (01FE) = 01). Notate anche che PC = 0102, l'indirizzo della prima istruzione della subroutine di ritardo.

Passo 4

Premete il tasto GO per proseguire nell'esecuzione del programma finchè non viene incontrata l'istruzione di RET alla locazione 010E. Controllate il registro SP prima e dopo l'esecuzione dell'istruzione di RET:

Prima: SP = __________ PC = __________
Dopo: SP = __________ PC = __________

Noi abbiamo osservato che prima SP = 01FD e PC = 010E, mentre dopo SP = 01FF e PC = 0123. I due byte in cima allo stack sono stati estratti ed inseriti nel registro PC, in modo che l'esecuzione può riprendere dall'istruzione (LD A,00) che segue l'istruzione di CALL DELAY.

Passo 5

Vediamo ora quando dobbiamo usare le subroutine. Per decidere se un gruppo di istruzioni potrebbe essere o meno una subroutine, bisogna seguire due criteri:

1. CRITERIO FUNZIONALE - Il set di istruzioni costituisce un'unità logica con ingressi ed uscite ben definiti? Cioè, ha senso separare la funzione svolta dal gruppo di istruzioni dal resto del programma?

2. CRITERIO DI EFFICIENZA - Il fatto di convertire un set di istruzioni in una subroutine costa in termini di tempo e di spazio di memoria, più di quanto sia giustificato dalle considerazioni funzionali del primo criterio?
   Prendiamo in considerazione la seguente "analisi del peggiore dei casi":

Supponiamo di dover decidere se un gruppo di istruzioni equivalente a M byte di memoria debba essere convertito o meno in una subroutine, e quindi richiamato dal programma di cui esso è attualmente una parte. Supponiamo inoltre che questo gruppo di istruzioni compaia R volte nel programma. Quindi,

Numero di byte usati quando il gruppo di istruzioni è inserito nel programma	= (n. volte nel programma) × (n. byte per ogni volta) = R×M
Massimo numero di byte usati quando il gruppo di istruzioni è utilizzato come subroutine	= (n. byte nella subroutine compreso RET) + (n. di chiamate nel programma) × (n. byte per ogni CALL) + (3 byte per definire lo stack) + (2 byte di stack per l'indirizzo di rientro) = M+ (n. di RET)+ 3R+ 3+ 2 = M+ (n. di RET) + 3R+ 5

Il criterio dell'efficienza di spazio viene soddisfatto finchè è:

$$R \times M > M + (\text{n. di RET}) + 3R + 5$$

Per quanto riguarda il criterio dell'efficienza rispetto al tempo di esecuzione, l'uso delle subroutine comporta sempre una perdita di velocità di esecuzione rispetto alla ripetizione

continua nel corso del programma dei gruppi di istruzioni. L'istruzione di CALL è una delle istruzioni più lente del set di istruzioni dello Z-80. La ragione di questo sta nel fatto che la CPU non solo deve leggere tre byte di memoria per interpretare l'istruzione e cambiare il registro PC, ma deve anche inserire il vecchio valore di PC nello stack.
L'istruzione di RET consuma del tempo perchè essa deve accedere allo stack per modificare il registro PC. Il fatto di dover aggiungere delle istruzioni di CALL e RET, per ogni chiamata di subroutine, fa sì che il programma diventi necessariamente più lento. Questo svantaggio è di solito compensato dalla minore occupazione di memoria che si può avere facendo uso di subroutine. In casi in cui i tempi sono critici, si possono usare le istruzioni di RST più veloci o, in casi estremi, si rinuncerà alla possibilità di usare subroutine in modo da rendere minimo il tempo di esecuzione.

Passo 6

Analizzate il programma e la subroutine riportati come esempio in questo esperimento per decidere se l'uso di una subroutine giustificato in termini di tempo e spazio. Giustificate la vostra risposta.

La nostra risposta è che, basandosi *strettamente* sul criterio di spazio, non è stata una buona idea usare la subroutine.*DELAY* è stata chiamata solo una volta, lo spazio di memoria occupata è minore per l'implementazione senza subroutine:

```
        LD B,03H
DELAY:  LD C,FFH
LOOP2:  LD D,FFH
LOOP3:  DEC D
        JR NZ,LOOP3
        DEC C
        JR NZ,LOOP2
        DJNZ DELAY
        LD A,00H
        RST 38H
```

Byte totali = 19
Risparmio: (n. byte usati nel programma e nella subroutine di esempio)
+ 2 byte dello stack per l'indirizzo di rientro
− 19
= 9 byte

Nota: 9 = (2 byte per lo stack) + (3 byte per LD SP, B3)
+ (3 byte per CALL DELAY) + (1 byte per RET).

L'utilizzo di subroutine, anzichè inserire direttamente nel programma le istruzioni desiderate, comporta sempre un sacrificio in termini di velocità di esecuzione. Comunque, un sacrificio per quanto riguarda la velocità o un sacrificio nell'occupazione della memoria (come sarebbe necessario nel suddetto esempio) è spesso compensato dai vantaggi che si ottengono dalla stesura di programmi modulari ben strutturati, che sono più facili da mettere a punto e più facili da mantenere e modificare.

Passo 7

Eseguite il programma di chiamata di subroutine modificando il valore del parametro di ritardo passato alla subroutine tramite il registro B.

ESPERIMENTO N. 3

Scopo

Questo esperimento illustra l'esecuzione dell'istruzione di RST N

Programma N. 22

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
0130	31 FF	LD SP,01FFH	;Posiziona lo stack del sistema
0133	01 00 01	LD BC,0100H	;BC = numero di byte di memoria da azzerare
0136	21 00 04	LD HL,0400H	;HL = indirizzo di partenza
0139	D7	RST 10H	;CALL della subroutine di azzeramento
013A	FF	RST 38H	;Restituisci il controllo al sistema operativo.
0010	F5	PUSH AF	;Salva quanto contenuto dai registri prima
0011	C5	PUSH BC	;della chiamata della subroutine
0012	D5	PUSH DE	
0013	E5	PUSH HL	
0014	36 00	LD (HL),00H	
			;Carica tutti zeri nella prima locazione
0016	54	LD D,H	;Sposta l'indirizzo contenuto nella coppia di ;registri HL nella coppia di registri DE.
0017	5D	LD E,L	
0018	13	INC DE	;Incrementa DE
0019	ED B0	LDIR	;Azzera tutte le locazioni a partire da DE
001B	06 03	LD B,03H	;Fissa il valore del ritardo per DELAY
001D	CD 02 01	CALL DELAY	;Chiama la routine Delay
0020	F1	POP HL	;Ripristina i contenuti originali dei registri,
0021	C1	POP DE	;prima della CALL
0022	D1	POP BC	
0023	E1	POP AF	
0024	C9	RET	

Passo 1

Caricate il programma e la subroutine agli indirizzi indicati. Assicuratevi che venga caricata in memoria anche la subroutine DELAY dell'Esperimento N. 2.

Passo 2

Il programma principale (indicato come *Programma*) usa le coppie di registri BC e HL per trasferire due parametri alla subroutine posta a 0010. BC contiene il numero di byte di memoria in cui bisogna caricare 00, HL contiene l'indirizzo del primo byte: la subroutine userà così l'istruzione di LDIR per azzerare le "BC" locazioni di memoria consecutive, partendo dalla locazione "HL". Notate che, per via del particolare indirizzo di partenza della subroutine, 0010,ci sono due istruzioni di chiamata che possono essere usate per la subroutine, cioè CALL 0010 e RST 16. L'istruzione di CALL 0010 occupa tre byte di memoria dato che il codice esadecimale ad essa relativo è CD 10 00. Il codice

esadecimale dell'istruzione RST 16 è D7, un solo byte. Entrambe le istruzioni hanno la stessa identica funzione.

La subroutine, di inizio 0010, usa una tecnica standard per mantenere i registri A,F,B,C,D,E,H e L esattamente come erano al momento della chiamata della subroutine. Le prime quattro istruzioni della subroutine *inseriscono* i contenuti dei registri nello stack. Le ultime quattro istruzioni prima del rientro, ripristinano di nuovo i registri usando istruzioni di POP.
State particolarmente attenti all'ordine nel quale le coppie di registri vengono inserite ed estratte. L'ordine di inserimento deve essere esattamente il contrario dell'ordine di estrazione. La ragione di questo sta nella struttura a LIFO (last-in-first-out) dello stack. Quando una subroutine non altera alcun registro, si dice che la subroutine *conserva lo stato della CPU*.
Quindi la subroutine a 0010 conserva costante lo stato della CPU, mentre la subroutine DELAY (alla locazione 0102) lo altera.
Notate che la subroutine a 0010 richiama lei stessa una subroutine, (istruzione di CALL DELAY alla locazione 001D). Si tratta della chiamata di una subroutine *nidificata*. Perciò, non ci sono regole che impediscono a delle subroutine di chiamare altre subroutine che, a loro volta, possono richiamarne altre ancora. Le subroutine possono perfino chiamare se stesse! La tecnica di *programmazione ricorsiva* la tecnica cioé secondo la quale le subroutine richiamano se stesse, è molto efficace, ma anche molto difficile da utilizzare: vi basti sapere che esiste.

Che cosa succede quando vengono chiamate delle subroutine in modo nidificato? In sostanza gli indirizzi di rientro continuano a venire inseriti nello stack, finchè una subroutine esegue finalmente una istruzione di RET, RET NZ, RET Z, RET C, RET NC, RET P, RET M, RET PE o RET PO, momento in cui un indirizzo viene estratto dallo stack. Eseguite il programma posto alla locazione 0130. Inserite in modo opportuno i punti d'interruzione (Breakpoint) del programma, in modo che possiate vedere lo stack che aumenta e diminuisce man mano che procedete nell'esecuzione.

Passo 3

Attualmente, la memoria a lettura-scrittura è occupata da un programma, due subroutine, uno stack, ed un blocco di memoria di dati di programma. Spesso, è un'ottima idea avere una mappa che mostri l'utilizzazione della memoria. La Fig. 8-6 vi presenta una mappa di memoria per questo esperimento.

Indirizzi		
00 00		
00 10 00 20 00 25	SUBROUTINE	Carica "BC" byte di memoria partendo dalla locazione HL (mantiene intatto lo stato della CPU)
00 30 01 00		
01 00 01 0D	SUBROUTINE DELAY	Ritardo che dipende dal contenuto del registro B

01 10 01 20			
01 30 01 3D	PROGRAMMA PRINCIPALE		Tutte le routine sono richiamate da questo programma
01 40			
01 E0 01 E8 01 F0 01 F8 01 FF 02 00	STACK		
04 00 05 00	DATI		Il blocco di memoria che deve essere caricato con zeri

Figura 8-6. Mappa di memoria

ESPERIMENTO N. 4

Scopo

Questo esperimento illustra quattro tecniche per trasferire dei parametri alle subroutine.

Tecnica N. 1: Trasferimento dei parametri tramite registri.

Programma N. 23

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
0200	31 FF 08	LD SP,08FFH	;Posiziona lo stack del sistema
0203	01 00 01	LD BC,0100H	;BC= N. byte di memoria da azzerare
0206	21 00 04	LD HL,0400H	;HL= indirizzo di partenza
0209	CD 10 02	CALL ZERO1	;Richiama la routine di azzeramento
020C	FF	RST 38H	;Restituisci il controllo al sistema operativo

Subroutine

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
0210	36 00	ZERO 1:	LD (HL),00H	;Azzera la prima locazione
0212	54		LD D,H	;Sposta l'indirizzo contenuto
0213	5D		LD E,L	;nella coppia di registri
				;HL, nella coppia di registri DE
0214	13		INC DE	;Incrementa DE
0215	ED B0		LDIR	;Azzera le locazioni partendo ;da DE
0217	C9		RET	;Restituisci il controllo al ;programma principale.

Tecnica N. 2: Trasferimento dei parametri tramite lo stack

Programma N. 24

0220	31 FF 08		LD SP,08FFH	
0223	01 00 01		LD BC,0100H	
0226	21 00 04		LD HL,0400H	
0229	C5		PUSH BC	;Inserisci i parametri nello stack
022A	E5		PUSH HL	
022B	CD 31 02		CALL ZERO2	
022E	FF	RST 38H		

Subroutine

0231	D1	ZERO 2:	POP DE	;Estrai l'indirizzo di rientro ;dallo stack
0232	E1		POP HL	;Estrai i parametri dallo stack
0233	C1		POP BC	
0234	D5		PUSH DE	;Inserisci l'indirizzo rientro di ;nuovo nello stack
0235	36 00		LD (HL),00H	
0237	54		LD D,H	
0238	5D		LD E,L	
0239	13		INC DE	
0231	ED B0		LDIR	
023C	C9		RET	

Tecnica N 3: Trasferimento dei parametri tramite locazioni di memoria.

Programma N. 25

0240	31 FF 08	LD SP,08FFH	
0243	01 00 01	LD BC,0100H	
0246	21 00 04	LD HL,0400H	
0249	ED 43 00 08	LD (0800H),BC	;Memorizza i parametri in locazioni ;di memoria

```
024D    22 02 08                 LD (0802H),HL
0250    CD 56 02                 CALL ZERO3
0253    FF                       RST 38H
```

Subroutine

```
0256    ED 4B 00 08   ZERO 3:    LD BC,(0800H)
025A    2A 02 08                 LD HL,(0802H)    ;Trasferisci i parametri situati in
                                                  ;memoria nei registri corretti
025D    36 00                    LD (HL),00H
025F    54                       LD D,H
0260    5D                       LD E,L
0261    13                       INC DE
0262    ED B0                    LDIR
0264    C9                       RET
```

Tecnica N. 4: Trasferimento dei parametri tramite le locazioni di memoria immediatamente successiva alla istruzione di CALL

Programma N. 26

```
0265    31 FF 08                 LD SP,08FFH
0268    CD 72 02                 CALL ZERO4
026B    00 01                    DEFW 0100H       ;DEFW è uno "pseudo operatore"
026D    00 04                    DEFW 0400H       ;implementato sull'assembler ASS-Z
026F    FF                       RST 38H          ;della SGS-ATES per permettere ai
                                                  ;dati di coesistere con le istruzioni
```

Nelle ultime pagine di questo capitolo troverete una spiegazione esauriente.

Subroutine

```
0272    DD E1         ZERO 4:    POP IX           ;IX=indirizzo del primo parametro
0274    DD 4E 00                 LD C,(IX)
0277    DD 46 01                 LD B,(IX+01H)    ;Memorizza i dati nei registri
027A    DD 6E 02                 LD L,(IX+02H)
027D    DD 66 03                 LD H,(IX+03H)
0280    11 04 00                 LD DE,0004H      ;Somma 0004 ad IX per ottenere
0283    DD 19                    ADD IX,DE        ;l'indirizzo di rientro della
                                                  ;subroutine
0285    DD E5                    PUSH IX          ;Inserisci l'indirizzo di rientro nello
                                                  ;stack
0287    36 00                    LD (HL),00H
0289    54                       LD D,H
028A    5D                       LD E,L
028B    13                       INC DE
028C    ED B0                    LDIR
028E    C9                       RET
```

Passo 1

Caricate tutti e quattro i programmi e le relative subroutine. Verificate di averli caricati correttamente.

Passo 2

Studiate tutte e quattro le tecniche di trasferimento dei parametri, per essere sicuri di aver capito il funzionamento di ognuna.
Notate che non è però corretto che un programma chiami una subroutine associata ad un altro programma. Infatti tra un programma e la subroutine corrispondente esiste uno stretto accoppiamento, dovuto al modo in cui viene effettuato il trasferimento dei parametri.
Confrontiamo ora le diverse soluzioni presentate.
Per prima cosa analizziamo il problema dell'occupazione della memoria.

Tecnica	Programma	Subroutine	Totale N. di Byte
1	15	8	23
2	17	12	29
3	22	15	37
4	13	27	40

Anche se la tecnica N. 1 è quella che occupa la quantità di memoria minore in questo gruppo di esempi, si potrebbero facilmente trovare altri esempi nei quali è la tecnica 4 che si dimostra la più efficiente. Ogni volta che viene richiamata la subroutine ZERO1, sono necessari 9 byte di spazio nel programma di chiamata per preparare i parametri e chiamare la subroutine. Preparare e chiamare ZERO4 occupa solo 7 byte. Il grosso numero di byte addizionali presenti in ZERO4 per implementare la tecnica più complicata è però un *Costo one-time,* cioè che si verifica una sola volta.
Più ZERO4 viene richiamata, più diventano meno significativi i byte addizionali della subroutine, e più diventano significativi i risparmi nella sequenza di chiamata.
Per usare la tecnica N. 1, deve essere possibile inserire tutti i parametri da trasferire nei registri disponibili. Qualche volta ciò può costituire un serio problema. Le tecniche N. 2 e 3 usano entrambe la memoria per il trasferimento dei parametri. La tecnica N. 2 è l'ideale per il trasferimento di parecchi parametri che vengono usati dalla subroutine in una particolare sequenza. La tecnica N. 3 è l'unico modo pratico di trasferire le matrici o lunghe stringhe di caratteri fra le routine.

Passo 3

Eseguite tutte le coppie programma-subroutine. State particolarmente attenti alla tecnica N. 4 che è più complicata delle altre.

ESPERIMENTO N. 5

Scopo

Scopo di questo esperimento è mostrare come si usano le tabelle di salto.

Programma N. 27

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
0900	61		DEFB 61H	;Valore N. 1
0901	41 09		DEFW 0941H	;Indirizzo per il Processo N. 1
0903	62		DEFB 62H	;Valore N. 2
0904	45 09		DEFW 0945H	;Indirizzo per il Processo N. 2
0906	63		DEFB 63H	;Valore N. 3
0907	50 09		DEFW 0950H	;Indirizzo per il Processo N. 3
0909	64		DEFB 64H	;Valore N. 4
090A	55 09		DEFW 0955H	;Indirizzo per il Processo N. 4
090C	00		DEFB 00H	;Indica la fine della ;tabella di salto
0915	21 FD 08	START:	LD HL,08FDH	;Inizializza HL
0918	23	NEXT:	INC HL	;Incrementa HL per puntare
0919	23		INC HL	;al primo valore della tabella di salto
091A	23		INC HL	
091B	7E		LD A,(HL)	;Carica il valore nell'accumulatore
091C	B7		OR A	;Setta il flag di zero se A è zero
091D	CA 38 00		JP Z, 0038H	;Se A=0, si è alla fine della tabella e quindi restituisci il controllo al sistema operativo
0920	B8		CP B	;B=A?
0921	20 F5		JR NZ,NEXT	;Se no, prova con il valore successivo ;nella tabella
0923	23		INC HL	;Se sì, le due posizioni successive ;contengono l'indirizzo della routine da ;eseguire
0924	5E		LD E,(HL)	;Memorizza l'indirizzo in DE
0925	23		INC HL	;Prima LO, poi HI
0926	56		LD D,(HL)	
0927	62		LD H,D	;Sposta il contenuto di DE in HL
0928	6B		LD L,E	
0929	E9		JP (HL)	;Salta all'indirizzo in HL

Passo 1

Parliamo prima delle nuove istruzioni che compaiono nel programma suddetto. Per capire DEFB o DEFW, esaminiamo attentamente il codice oggetto relativo a queste istruzioni. Per esempio, DEFB 41H ha come codice esadecimale 41. In generale

DEFB <B1>

ha <B1> come codice esadecimale. DEFB sta per "DEFine Byte" (definisci il byte) perché tutto quello che l'istruzone fa, è indicare un valore (byte) che viene posto direttamente in memoria. DW ha un effetto analogo per gli indirizzi a due byte.

DEFW <B2> <B1>

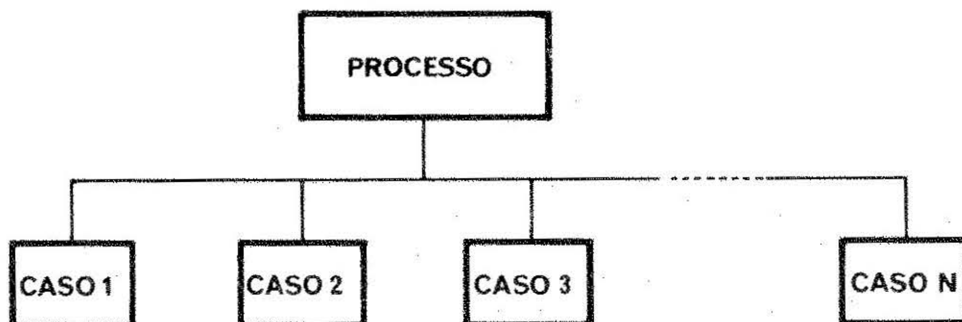
ha <B1> <B2> come codice esadecimale. DEFW sta per "DEFine Word" (definisci la parola), dove la parola è in questo caso un indirizzo assoluto a due byte.
E' importante notare la differenza esistente fra le istruzioni DEFB e DEFW e tutte le altre istruzioni di cui abbiamo parlato finora. Queste ultime corrispondono ad operazioni eseguite dalla CPU dello Z-80. DEFB e DEFW *non* corrispondono ad operazioni eseguite dallo Z-80, ma piuttosto danno luogo ad un inserimento diretto di dati in memoria. In questo modo, il programma assembler (o la persona che assembla manualmente), introduce nella memoria i valori desiderati.
Le istruzioni DEFB e DEFW sopra descritte sono chiamate *PSEUDO-ISTRUZIONI* del linguaggio assembler. Ciò è dovuto al fatto che esse non sono istruzioni eseguibili dallo Z-80. Si tratta invece di istruzioni che *l'ASSEMBLER* esegue quando genera il codice oggetto.
Un'altra pseudo-istruzione comune a molti assembler è quella di DEFine Storage (definisci l'area di immagazzinamento), il cui formato è:

DEFS n.

La pseudo-istruzione DEFS è usata per dire all'assembler di riservare un numero specofico, n, di byte nel codice oggetto come area di memoria lettura/scrittura.
L'assembler non inserisce alcun valore particolare nello spazio riservato al deposito, ma si limita a saltare n byte successivi prima di continuare a caricare in memoria il codice oggetto generato.
Il numero n può essere un numero esadecimale (seguito da un "H") o un numero decimale (seguito da un punto ".").
Dell'istruzione CP B alla locazione 09 20 parleremo dettagliatamente in un altro capitolo.
Il codice esadecimale relativo a questa istruzione è B8. L'istruzione confronta il contenuto del registro B con quello del registro A. Se il contenuto è uguale, il flag di zero viene posto a 1, altrimenti viene resettato. Perciò il programma di questo esperimento usa le istruzioni di CP b e JR NZ per determinare se A e B sono uguali e saltare in una delle due direzioni, a seconda del risultato.
Caricate il programma alla locazione indicata.

Passo 2

Una tabella di salto, altrimenti detta JUMP TABLE, è un sistema molto efficace per effettuare dei salti ove si abbia una situazione di questo tipo:



Ovvero, un processo termina con un certo risultato: a seconda di tale risultato viene eseguito quindi uno dei nuovi processi possibili (N nel caso di prima, 4 nella tabella di salto riportata nel programma all'inizio di questo esperimento). Il termine software per definire quanto detto è *ANALISI DEI CASI.*
Ogni possibile risultato costituisce un CASO; ovvero, nel CASO il risultato sia X, si attiva il processo J corrispondente.
La tabella di salto che vi abbiamo mostrato come esempio ha il seguente formato generale:

VALORE 1

INDIRIZZO 1

VALORE 2

INDIRIZZO 2

VALORE 3

INDIRIZZO 3

.

.

.

VALORE N

INDIRIZZO N

00

Sono elencati ogni possibile risultato e, subito dopo, l'indirizzo della routine ad esso relativa. 00 sta ad indicare la fine della tabella di salto. Per effettuare l'analisi dei casi, il risultato (nel registro B) viene confrontato (CP B) con ogni *valore* della tabella, finché si trova una corrispondenza o finché si arriva alla fine della tabella stessa. Se si trova una corrispondenza, la coppia di registri HL viene caricata con l'indirizzo che è posto subito dopo il byte del *valore*, e viene effettuato un salto a tale indirizzo.

Introducete dei rientri (FF) nel sistema operativo nelle diverse locazioni puntate dagli indirizzi in tabella.

Inserite nel registro B dei valori come 45,55 e 41 che si trovano nella tabella di salto, e 01, o 09 che non sono nella tabella, e eseguite ogni volta il programma passo passo partendo da 0915. Che cosa succede se iniziate l'esecuzione a 0900?

In tal caso eseguireste come istruzioni i vostri dati. Questa non è quasi mai una buona idea. Ricordatevi la differenza fondamentale, fra dati e codice eseguibile.

Passo 3

Esistono dei metodi alternativi a quello che vi abbiamo già presentato per implementare le tabelle di salto. Un metodo comune ed efficiente è quello di isolare i valori e gli indirizzi in due tabelle separate:

Valore 1	Indirizzo 1
Valore 2	Indirizzo 2
Valore 3	Indirizzo 3
⋮	⋮
Valore N	Indirizzo N

Viene prima cercata una corrispondenza di valori. Se tale corrispondenza risulta al primo valore, il controllo viene passato alla routine al primo indirizzo della tabella indirizzi. Come esercizio, cercate di implementare questo metodo per l'analisi dei casi. Ecco alcuni spunti:

1. Usate 00 per contrassegnare la fine di ogni tabella. Notate che se 00 è un possibile valore o byte d'indirizzo, dovete scegliere un simbolo nuovo per indicare la fine; o un nuovo metodo per rivelare la fine della tabella.
2. Notate che la tabella dei valori ha elementi di un byte, ma quella degli indirizzi ha elementi a due byte.
3. Come nel programma-tipo, supponete che il valore con cui si deve trovare una corrispondenza sia già in uno dei registri della CPU (scegliete quello più conveniente).
4. Usate le istruzioni di DEFB e DEFW per caricare le tabelle.

Una volta che avete scritto il programma, caricatelo ed eseguitelo.

CAPITOLO 9

ISTRUZIONI LOGICHE

INTRODUZIONE

La Tabella 9-1 contiene le istruzioni aritmetiche e logiche dello Z-80. La Tabella 9-2 contiene le istruzioni dello Z-80 che manipolano l'accumulatore ed i flag, cioè le istruzioni AF non specializzate. Di tutte queste istruzioni parleremo in questo Capitolo e nel Capitolo 11. In questo Capitolo vi presenteremo le istruzioni logiche, che eseguono operazioni logiche su parole binarie a otto bit. I codici mnemonici di queste istruzioni sono AND, XOR, OR e CPL. Dato che è necessario aver capito molto bene le operazioni logiche a più bit, per poter usare queste istruzioni in modo efficace, abbiamo incluso a tale riguardo anche una parte introduttiva.

OBIETTIVI

Alla fine di questo capitolo, sarete in grado di:

- Riassumere le tabelle della verità per le operazioni logiche a un bit AND, OR e XOR.
- Elencare gli esatti simboli Booleani per AND, OR, XOR e NOT.
- Spiegare come vengono eseguite le operazioni logiche su più bit.
- Eseguire le operazioni logiche di AND, OR, XOR su una coppia di dati a otto bit. L'operazione NOT è eseguita su dati di un byte.
- Scrivere il teorema di De Morgan in algebra Booleana.
- Spiegare il teorema di De Morgan usando i simboli logici.
- Elencare le istruzioni logiche comprese nel set di istruzioni dello Z-80.
- Spiegare come le istruzioni logiche possono essere usate in un programma di un microcomputer.
- Dare la definizione di *"maschera"*.

SORGENTE

	INDIRIZZAMENTO TRA REGISTRI							INDIR. TRAM. REG.	INDICIZZ.		IMM.
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
SOMMA 'ADD'	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	**C6 n**
SOMMA CON RIPORTO 'ADC'	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	**CE n**
SOTTRAZIONE 'SUB'	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	**D6 n**
SOTTRAZIONE CON RIPORTO 'SBC'	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	**DE n**
'AND'	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	**E6 n**
'XOR'	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	**EE n**
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	**F6 n**
CONFRONTO 'CP'	BF	B8	B9	BA	BB	BC	BD	BE	DD 3E d	FD 8E d	**FE n**
INCREMENTO 'INC'	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DECREMENTO 'DEC'	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

Tabella 9-1. Istruzioni aritmetiche e logiche a 8 bit.

Aggiustamento Decimale dell'Accumulatore 'DAA'	**27**
Complemento dell'Accumulatore "CPL"	**2F**
Negazione dell'Accumulatore 'NEG' (complemento a 2)	ED 44
Complemento del Flag di Riporto 'CCF'	**3F**
Set del Flag di Riporto 'SCF'	**37**

Tabella 9-2. Istruzioni AF non specializzate.

CHE COSA E' UN'ISTRUZIONE LOGICA

In questo paragrafo ci occuperemo delle operazioni logiche AND, OR, XOR (OR Esclusivo), e CPL (complemento). Un'operazione logica su due byte, è un'istruzione logica che viene eseguita tra due byte di dati, dove i bit corrispondenti di ogni byte vengono sottoposti bit per bit ad un'operazione logica AND, OR, o XOR. Nel microprocessore Z-80, uno dei byte di dati è presente nell'accumulatore, ed il risultato finale è pure posto nell'*accumulatore*. Questa è la ragione per cui l'accumulatore si chiama così; esso accumula i risultati finali delle operazioni logiche e aritmetiche.
L'istruzione logica ad un byte CPL opera direttamente sull'accumulatore e non viene coinvolto nessun altro registro o locazione di memoria.

Tabella della verità per operazioni logiche ad un bit

AND

B	A	Q
0	0	0
0	1	0
1	0	0
1	1	1

OR

B	A	Q
0	0	0
0	1	1
1	0	1
1	1	1

XOR

A		Q
0	0	0
0	1	1
1	0	1
1	1	0

NOT

A	Q
0	1
1	0

Nota: A e B sono i dati su cui si opera. Q è il risultato dell'operazione.

Queste tabelle sono dette "tabelle ad un bit" perchè i dati, A e B, sono ciascuno formati da un solo bit. XOR è un'abbreviazione di OR Esclusivo.

ALGEBRA BOOLEANA

Quando si parla di istruzioni logiche, è utile usare i simboli Booleani, i quali traggono origine dall'algebra Booleana, che è la matematica dei sistemi logici. I simboli dell'alfabeto come A, B, C, ... Q sono usati per rappresentare le variabili logiche e i simboli 1 e 0 sono utilizzati per rappresentare i diversi stati logici. Questa matematica ebbe origine in Inghilterra nel 1847, per opera di George Boole, ma non entrò nell'uso comune fino al 1938, quando Claude Shannon l'impiegò per analizzare i circuiti a relè delle reti telefoniche.

Per quanto riguarda l'algebra di Boole voi dovrete imparare solo i simboli base che vengono impiegati nei calcoli e che sono quindi in tutta la logica digitale. Fra questi simboli vi sono i seguenti:

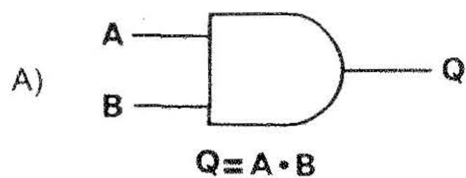
$+$ che indica un'operazione di *Addizione logica* e viene chiamato OR

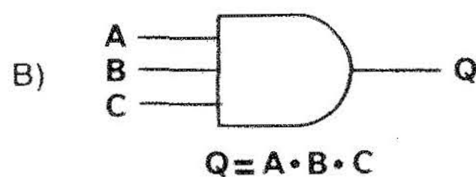
$\bullet$ che indica un'operazione di *Moltiplicazione logica* e viene chiamato AND

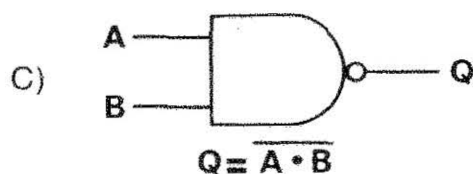
$\oplus$ che viene chiamato *OR Esclusivo* o XOR

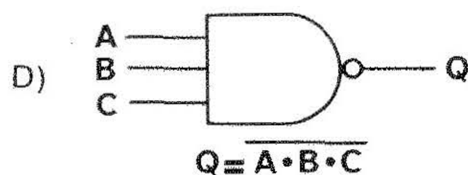
— che indica un'operazione di *Complemento logico* e viene chiamato NOT

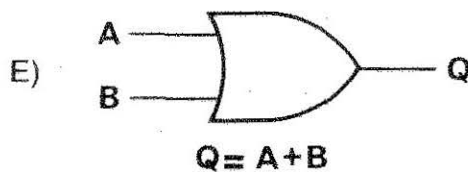
Il simbolo di complemento (negazione) è un trattino sopra la variabile logica quale A,B,C, ... Q. Allora, l'equazione Booleana per una porta AND a 2 ingressi è $Q = A \cdot B$ o, più semplicemente, $Q = AB$. Qui di seguito sono mostrate le equazioni Booleane per i diversi tipi di porte. *Fate attenzione all'uso del trattino,* (—), *per le porte* NAND *e* NOR.

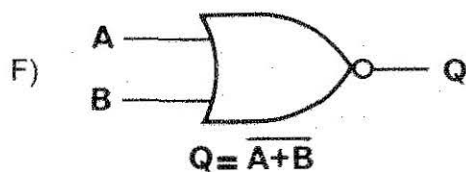


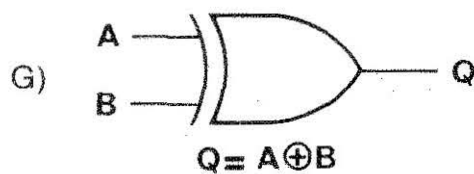












E' utile riassumere le operazioni simboliche per le quattro operazioni logiche che stiamo analizzando:

AND	OR	XOR	NOT
$0 \cdot 0 = 0$	$0 + 0 = 0$	$0 + 0 = 0$	$\overline{0} = 1$
$0 \cdot 1 = 0$	$0 + 1 = 1$	$0 + 1 = 1$	$\overline{1} = 0$
$1 \cdot 0 = 0$	$1 + 0 = 1$	$1 + 0 = 1$	
$1 \cdot 1 = 1$	$1 + 1 = 1$	$1 + 1 = 0$	

Queste sono operazioni logiche ad un bit.

OPERAZIONI LOGICHE A PIU' BIT

Le operazioni logiche a più bit sono eseguite nello stesso modo delle operazioni ad un bit, e quindi non è necessario introdurre dei nuovi concetti. Ogni bit di una parola binaria viene fatto operare logicamente sul corrispondente bit della seconda parola binaria producendo un risultato a più bit.

Considerate una variabile logica A a 8 bit; i diversi bit degli 8 che compongono la parola possono essere indicati come: A7, A6, A5, A4, A3, A2, A1 e A0, dove A0 è il bit meno significativo (il bit corrispondente al peso 2^0) e dove il bit A7 è quello più significativo (quello corrispondente al peso 2^7). In modo del tutto analogo considerate la variabile logica B a 8 bit, composta dai bit B7, B6, B5, B4, B3, B2, B1 e B0, con B0 bit meno significativo e B7 bit più significativo.
Allora l'operazione logica, $A \cdot B = Q$, significa:

$$
\begin{aligned}
A0 \cdot B0 &= Q0 \\
A1 \cdot B1 &= Q1 \\
A2 \cdot B2 &= Q2 \\
A3 \cdot B3 &= Q3 \\
A4 \cdot B4 &= Q4 \\
A5 \cdot B5 &= Q5 \\
A6 \cdot B6 &= Q6 \\
A7 \cdot B7 &= Q7
\end{aligned}
$$

Il risultato dell'operazione logica è la variabile logica Q che ha Q0 come bit meno significativo e Q7 come bit più significativo. In altre parole, *le operazioni su più bit vengono eseguite bit per bit effettuando una serie di operazioni logiche a un bit.*
E' più semplice eseguire le operazioni logiche se le parole binarie a più bit sono messe una sopra l'altra; quindi se A=11110000 e B=00111100 allora A·B:

```
11110000
00111100
--------
00110000
```

cioè Q=0011000. Abbiamo eseguito l'operazione logica AND e, per ottenere il risultato finale, abbiamo usato le relazioni $0 \cdot 1 = 0$ e $1 \cdot 1 = 1$. In modo analogo, l'operazione logica a più bit, A+B=Q ha il seguente significato

$$
\begin{aligned}
A0 + B0 &= Q0 \\
A1 + B1 &= Q1 \\
A2 + B2 &= Q2 \\
A3 + B3 &= Q3 \\
A4 + B4 &= Q4 \\
A5 + B5 &= Q5 \\
A6 + B6 &= Q6 \\
A7 + B7 &= Q7
\end{aligned}
$$

Ancora una volta il risultato dell'operazione logica è la variabile logica Q a 8 bit tale che, se A=11110000 e B=00111100 allora Q=A+ B diventa,

```
11110000
00111100
--------
11111100
```

cioé Q=11111100. Abbiamo eseguito l'operazione logica OR e, per ottenere il risultato finale, abbiamo usato le relazioni $0 + 1 = 1$, $1 + 1 = 1$ e $0 + 0 = 0$. *Notate che il simbolo "+" rappresenta l'operazione logica OR e anche il "più" delle operazioni aritmetiche.*

Questo vi puo confondere un po', prestate quindi attenzione.
L'ultima operazione logica che ci interessa è $A \oplus B = Q$ che ha il seguente significato

$$
\begin{aligned}
A0 \oplus B0 &= Q0 \\
A1 \oplus B1 &= Q1 \\
A2 \oplus B2 &= Q2 \\
A3 \oplus B3 &= Q3 \\
A4 \oplus B4 &= Q4 \\
A5 \oplus B5 &= Q5 \\
A6 \oplus B6 &= Q6 \\
A7 \oplus B7 &= Q7
\end{aligned}
$$

Il risultato di questa operazione di OR Esclusivo è una variabile logica Q a 8 bit, tale che se A=11110000 e B=00111100, allora Q=A + B diventa

$$
\begin{array}{r}
11110000 \\
\underline{00111100} \\
11001100
\end{array}
$$

cioè Q=11001100. Abbiamo eseguito un OR Esclusivo e, per ottenere il risultato finale, abbiamo usato le relazioni $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, e $1 \oplus 1 = 0$.

NOT

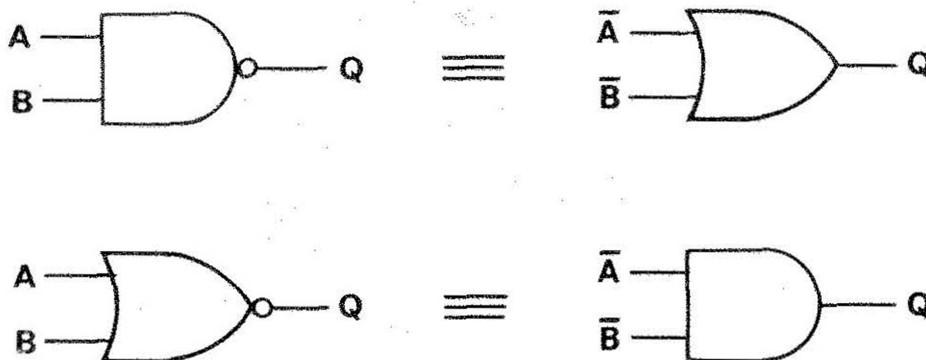
L'operazione logica NOT effettua il complemento di un bit o di gruppi di bit. Se A = 11110000, allora Q = 00001111.

TEOREMA DI DE MORGAN

Un importante teorema dell'algebra Booleana è il teorema di De Morgan, che può essere rappresentato in due modi:

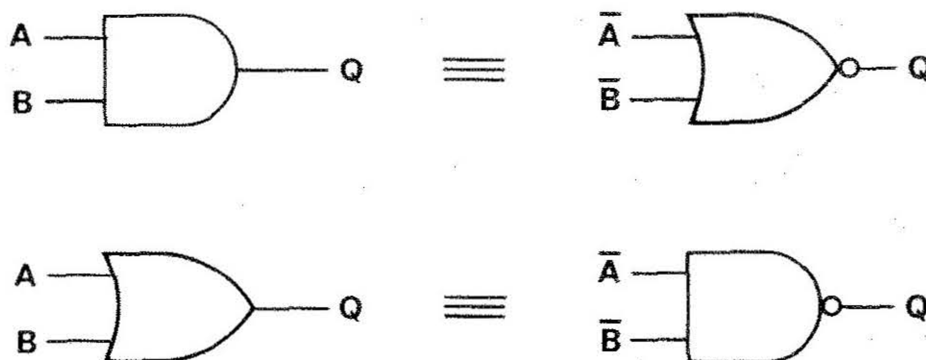
$$\overline{A \cdot B} = \overline{A} + \overline{B}$$
$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

Un modo più interessante di enunciare il teorema di De Morgan è quello che fa uso dei simboli logici, cioè:



E' un risultato molto importante e vi troverete ad usarlo molto spesso nell'elettronica digitale e nell'interfacciamento dei microcomputer. Secondo tale teorema è possibile ottenere una funzione NOR negando tutti gli ingressi di una porta AND; e, d'altro canto, ottenere una funzione NAND negando tutti gli ingressi di una porta OR.

Il teorema di De Morgan può essere rappresentato anche dai seguenti simboli logici:



Questi simboli dimostrano che potete ottenere una funzione AND negando tutti gli ingressi di una porta NOR; e, d'altro canto, ottenere una funzione OR negando tutti gli ingressi di una porta NAND. I circuiti integrati contenenti una porta NAND sono molto comuni e poco costosi.
Il teorema di De Morgan vi mostra come poter creare delle porte OR e NOR usando delle porte NAND. Delle porte NAND, NOR, AND e OR parleremo dettagliatamente nel Vol. 2.

GRUPPO DI ISTRUZIONI LOGICHE DELLO Z-80

Tutte le operazioni logiche di cui abbiamo parlato vengono implementate dalla CPU dello Z-80 come istruzioni appartenenti al gruppo di istruzioni logiche a 8 bit. Vediamo queste istruzioni più da vicino. State particolarmente attenti al modo in cui queste istruzioni influenzano il registro F dei flag, dato che questo aspetto dell'operazione è spesso essenziale al fine di un utilizzo efficace delle istruzioni stesse.

Complementa l'accumulatore: CPL

Complementare l'accumulatore significa eseguire un'operazione di NOT sul byte contenuto nell'accumulatore. Questa istruzione ad un solo byte ha come codice esadecimale 2F e come codice mnemonico CPL. I flag di carry, di zero, P/V e di segno non sono influenzati da questa istruzione. Per esempio:

	Contenuto dell'accumulatore
Prima dell'esecuzione di CPL	10111010
Dopo l'esecuzione di CPL	01000101

Ecco una buona applicazione dell'istruzione di CPL:

```
CPL A
INC A
```

Queste due istruzioni complementano a due il contenuto dell'accumulatore.

AND con l'accumulatore: AND

Le undici diverse istruzioni di AND, comprese nel set di istruzioni dello Z-80, hanno come codice mnemonico generale AND S, in cui S dipende dal metodo d'indirizzamento. Il flag di carry viene resettato da questa istruzione e sia il flag di segno che il flag di zero sono alterati coerentemente con il risultato dell'operazione. Il flag P/V rispecchia le parità del risultato, e viene settato se la parità (numero dei bit a 1) è pari, e in caso contrario, resettato.

Ad esempio, prendiamo in considerazione l'esecuzione dell'istruzione AND B. Questa istruzione fa sì che lo Z-80 esegua un'operazione di AND logico fra il contenuto del registro B e il contenuto dell'accumulatore. Per esempio:

	Accumulatore	Registro-B	Flag-S	Flag-Z	Flag-P/V
Prima della esecuzione di AND B	11001100	10001011	X	X	X
Dopo l'esecuzione di AND B	10001000	10001011	1	0	1

(X = non significativo)

Notate che, anche se potrebbe sembrare che l'istruzione di AND A, che effettua l'AND logico dell'accumulatore con se stesso, sia un'istruzione inutile, in realtà non lo è. Come risultato di questa istruzione, vengono alterati sia il flag zero che il flag di segno.
AND A setta a uno il flag di zero se e solo se il contenuto dell'accumulatore è 00. AND A setta il flag di segno se e solo se l'accumulatore contiene un numero in complemento a due negativo.
Per esempio:

	Accumulatore	Flag-S	Flag-Z	Flag-P/V
Prima della esecuzione di AND A	10000001	X	X	X
Dopo l'esecuzione di AND A	10000001	1	0	1

(X = non significativo)

L'istruzione di AND viene applicata utilmente anche per implementare una tecnica chiamata di "mascheratura". Possiamo così definirla:

Mascheratura Tecnica logica mediante la quale certi bit di una parola di più bit vengono annullati o "coperti"

Una maschera per il viso copre alcune parti della faccia. Allo stesso modo, una maschera, usata in un'operazione da un computer, copre alcuni dei bit di una parola a più bit, lasciando "scoperti" solo quei bit che sono importanti per la prosecuzione del programma. Prendiamo in considerazione la seguente sequenza di istruzioni

```
LD A,(0F32H)
AND 01H
```

Il flag di zero è settato o resettato a seconda del valore del bit meno significativo nel byte della locazione di memoria 0F32.
Questa sequenza di istruzioni si serve di un'operazione di mascheratura per controllare il bit D0 di un byte di memoria, essendo la maschera il byte 01 dell'istruzione di AND.
Analogamente, si può usare la stessa tecnica per controllare altri bit. Si può usare la maschera anche per analizzare dei gruppi di bit all'interno di un byte. Per esempio, la sequenza di istruzioni:

```
LD A,(0F32H)
AND 0FH
```

azzera i quattro bit più significativi e lascia invariati i quattro bit meno significativi dell'accumulatore, permettendo ad un programma di vedere solo la parte meno significativa del byte contenuto in 0F32.

OR Esclusivo con l'Accumulatore: XOR

Il codice mnemonico generale dell'istruzione di OR Esclusivo è XOR S, in cui S dipende dal metodo d'indirizzamento.

Come per l'istruzione di AND, il flag di carry viene resettato e i flag di segno, di zero e di parità/overflow sono alterati in base al valore del risultato dall'operazione.

Per esempio, osserviamo l'istruzione di XOR (HL) con HL = 1AB6H, che fa in modo che lo Z-80 esegua un'operazione di OR Esclusivo fra il contenuto dell'accumulatore e il contenuto della locazione di memoria 1AB6 puntata dalla coppia di registri HL, e lascia il risultato nell'accumulatore.

	Accumulatore	HL	(HL)	Flag-S	Flag-Z	Flag-P/V
Prima dell'esecuzione	10010001	1AB6	00110000	X	X	X
Dopo l'esecuzione	10100001	1AB6	00110000	1	0	0

Quindi, eseguendo un'operazione di XOR tra il byte 00 e il contenuto dell'accumulatore, si lascia invariato l'accumulatore, ma vengono alterati i flag di segno e di zero. L'istruzione di XOR A azzera l'accumulatore tramite un'istruzione ad un byte, ed è perciò preferibile a LD A,00H. Eseguendo un'operazione XOR tra FF e l'accumulatore, si ottiene lo stesso effetto ottenuto dall'istruzione di CPL, a parte il fatto che vengono alterati i flag di carry, di zero e di segno.

OR con l'accumulatore: OR

Le undici istruzioni di OR hanno come codice mnemonico generale OR S, in cui S dipende dal metodo d'indirizzamento. I flag sono alterati dall'istruzione di OR nello stesso modo in cui lo sono nel caso delle istruzioni di XOR e AND. Ovvero, il flag di carry è resettato e i flag di segno, di zero e di parità/overflow sono alterati a seconda del risultato dell'operazione. Per esempio, osserviamo l'istruzione di OR (IX + 02H).

Questa istruzione fa sì che lo Z-80 esegua un'operazione di OR logico fra il contenuto dell'accumulatore e il contenuto della locazione di memoria posta due byte al disopra della locazione indirizzata dal registro IX.

	Accumulatore	IX	(IX+02H)	Flag-S	Flag-Z	Flag-P/V
Prima dell'esecuzione	00000011	1AB4	00110010	X	X	X
Dopo l'esecuzione	00110011	1AB4	00110010	0	0	1

Quindi l'istruzione di OR non puo essere usata per azzerare l'accumulatore. Come avete già visto in molti esperimenti, OR A si può usare per verificare se l'accumulatore è zero. Si può benissimo usare un'istruzione di OR anche per determinare se una coppia di registri è zero. Per esempio, la seguente sequenza di istruzioni verifica se DE=0000:

```
LD A,D
OR E
```

L'unico caso in cui l'operazione di OR E può lasciare il flag di zero settato è che sia D che E siano uguali a 00. Avete visto applicare questa tecnica alla coppia di registri BC nell'ultimo esperimento del Capitolo 6. Il motivo per cui questa tecnica è così utile è che le istruzioni di incremento e di decremento a 16 bit *NON* alterano nessuno dei flag. Perciò, se una coppia di registri viene usata come contatore di loop e se il loop termina quando la coppia di registri contiene 0000, il flag di zero deve essere settato in modo esplicito da programma, ad esempio utilizzando la sequenza sopra riportata.

ISTRUZIONI LOGICHE E CONTROLLO DEI DISPOSITIVI ESTERNI

Le istruzioni logiche vi permettono di determinare se un dispositivo esterno è acceso (on) o spento (off), oppure se sono verificati o meno degli eventi particolari. Per esempio, supponiamo che si usi lo 0 logico e l'1 logico per rappresentare una delle seguenti operazioni:

A. Stato di un dispositivo
   0 = dispositivo off
   1 = dispositivo on

B. Verificarsi di un evento
   0 = l'evento non si è verificato
   1 = l'evento si è verificato

In un capitolo successivo, imparerete come usare l'istruzione IN per introdurre da un dispositivo esterno otto bit di dato nell'accumulatore, e imparerete anche come usare ogni singolo bit per rappresentare lo stato on/off di un dispositivo, oppure il verificarsi o il non verificarsi di un evento.
Con otto bit potete quindi rappresentare otto diversi dispositivi o eventi. Considerate i seguenti otto dispositivi che possono essere accesi (on) o spenti (off):

BIT 0: Dispositivo misuratore di pressione
BIT 1: Dispositivo misuratore di temperatura
BIT 2: Dispositivo misuratore di velocità
BIT 3: Dispositivo misuratore di portata
BIT 4: Dispositivo misuratore di tensione
BIT 5: Dispositivo misuratore di corrente
BIT 6: Dispositivo d'ingresso del terminale ASCII (tastiera)
BIT 7: Dispositivo di uscita del terminale ASCII (stampante o CRT)

Questi otto dispositivi sono associati a otto bit che possono essere letti tutti insieme e trasferiti nell'accumulatore del microprocessor Z-80. Una volta che sono memorizzati, potete usare le istruzioni logiche per determinare quali dispositivi siano on e quali siano off.

INTRODUZIONE AGLI ESPERIMENTI

Gli esperimenti che seguono hanno lo scopo di illustrare quello che avete imparato nel Capitolo 9 sulle istruzioni logiche.
Gli esperimenti che eseguirete si possono così riassumere:

Esperimento	Commenti
1.	Illustra una routine AND a 16 bit
2.	Illustra l'uso delle istruzioni AND, CPL e XOR applicate al controllo dei dispositivi. Queste istruzioni vengono usate per determinare se un dispositivo ha cambiato stato e, in caso affermativo, quale è stata la variazione cioè se da on a off o se da off a on.

ESPERIMENTO N. 1

Scopo

Questo esperimento illustra un programma che esegue un'operazione di AND logico fra il contenuto delle coppie di registri BC e DE e carica il risultato in HL.

Programma N. 28

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
0200	78	AND16:	LD A,B	
0201	A2		AND D	;Esegui un'operazione di AND sugli 8 bit più ;significativi
0202	67		LD H,A	;Carica il risultato nel registro H
0203	79		LD A,C	
0204	A3		AND E	;Esegui un'operazione di AND sugli 8 bit ;meno significativi
0205	6F		LD L,A	;Carica il risultato nel registro L
0206	B4		OR H	;Setta il flag di zero se il contenuto di HL è 00, ;altrimenti resetta il flag di zero.
0207	FF		RST 38H	

Passo 1

Caricate il programma all'indirizzo indicato. Inizializzate le coppie di registri BC e DE ai due valori che vi mostriamo qui di seguito e quindi eseguite il programma. Scrivete le vostre osservazioni sul contenuto della coppia di registri HL e sui flag di zero, di segno, di P/V e di carry.

Notate che il flag di zero è il bit D6 del registro F, il flag di segno è il bit D7, il flag P/V è il bit D2, e il flag di carry (riporto) è il bit D0 secondo la figura di seguito riportata:

7	6	5	4	3	2	1	0
S	Z	X	H	X	P/V	N	C

BC=0 0 1 1 0 0 1 1   1 1 0 0 1 0 0 1   o 33 C9
DE=1 1 1 1 0 0 0 0   0 1 1 1 0 0 1 1   o F0 73
HL=

Flag di Zero=   Flag di Segno=   Flag di P/V=   Flag di Riporto=

Noi abbiamo osservato che HL=30 41 e che i flag erano settati in questo modo:

Flag di Zero=0   Flag di Segno=0   Flag di P/V=1   Flag di Riporto=0

BC= 1 0 1 0 0 1 0 1   1 0 1 0 0 1 0 1   o A5 A5
DE= 1 1 0 0 1 1 0 0   0 1 0 1 1 1 1 1   o CC 5F
HL=

Flag di Zero=   Flag di Segno=   Flag di P/V=   Flag di Riporto=

Noi abbiamo osservato che HL= 84 05 e che i flag erano settati in questo modo:

Flag di Zero=0   Flag di Segno=1   Flag di P/V=0   Flag di Riporto=0

BC= 1 1 1 1 1 1 1 1   0 0 0 0 0 0 0 0   o FF 00
DE= 0 0 0 0 0 0 0 0   1 1 1 1 1 1 1 1   o 00 FF
HL=

Flag di Zero= Flag di Segno= Flag di P/V= Flag di Riporto=

Noi abbiamo osservato che HL= 00 00 e che i flag erano settati in questo modo:

Flag di Zero=1 Flag di Segno=0 Flag di P/V=1 Flag di Riporto=0

Passo 2

Scrivete, caricate e verificate un programma che esegue un'operazione di OR a 16 bit fra le coppie di registri BC e DE; il risultato sia posto in HL e lo stato del flag Z sia coerente col risultato. Controllate il programma per verificare la correttezza caricando dei valori di prova in BC e DE e controllando la corrispondenza tra i valori previsti (per HL e per il flag di zero) e i valori osservati.

ESPERIMENTO N. 2

Scopo

Scopo di questo esperimento è quello di rilevare quale, tra otto dispositivi, abbia cambiato stato nell'intervallo di tempo che intercorre tra due letture e in quale direzione è avvenuto tale cambiamento, (da on in off o da off in on). Eccovi l'elenco degli otto dispositivi, con il numero del bit ad essi associato:

BIT 0: Dispositivo misuratore di pressione
BIT 1: Dispositivo misuratore di temperatura
BIT 2: Dispositivo misuratore di velocità
BIT 3: Dispositivo misuratore di portata
BIT 4: Dispositivo misuratore di tensione
BIT 5: Dispositivo misuratore di corrente
BIT 6: Dispositivo misuratore di livello di liquidi
BIT 7: Dispositivo misuratore di frequenza

Uno stato logico 1 indica che il dispositivo è in ON, e lo stato logico 0 indica che il dispositivo è in OFF.
L'ingresso dei due byte di stato (quello precedente e quello attuale) è simulato con le due istruzioni LD r, <B2>.

Programma N. 29

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
0100	06 88	LD B,88H	;Simula l'ingresso del byte di stato ;precedente nel registro B
0102	3E 09	LD A,09H	;Simula l'ingresso del byte di stato ;corrente nell'accumulatore
0104	4F	LD C,A	;Copia lo stato attuale nel registro C
0105	A8	XOR B	;Esegui un OR esclusivo fra il contenuto ;di A e di B. Nel risultato lo stato logico 1 ;contrassegna i dispositivi il cui stato è ;cambiato
0106	57	LD D,A	;Conserva questa informazione nel ;registro D
0107	A0	AND B	;Esegui un'operazione di AND fra il ;contenuto di A e di B. Nel risultato, ;i bit a 1 contrassegnano i dispositivi il ;cui stato è cambiato da on in off

0108	67	LD H,A	;Conserva questa informazione nel ;registro H
0109	2F	CPL	;Complementa l'accumulatore. Nel ;risultato, i bit a 1 indicano un ;dispositivo che non ha cambiato stato ;da on ad off, cioè un dispositivo che è ;rimasto invariato o che ha cambiato stato ;da off a on
010A	A2	AND D	;Esegui un'operazione di AND fra il ;contenuto di A e di D. Nel risultato, lo ;stato logico 1 contrassegna un dispositivo ;il cui stato è cambiato da off a on
010B	6F	LD L,A	;Conserva questa informazione nel registro L
010C	FF	RST 38H	;Restituisci il controllo al sistema ;operativo.

Passo 1

Caricate il programma nelle locazioni di memoria indicate. Verificate di averlo caricato correttamente.

Passo 2

Esaminate attentamente il programma per verificare i seguenti risultati:

a. Se il bit n nel registro D è ad 1, il dispositivo ad esso associato ha cambiato stato.
b. Se il bit n nel registro L è ad 1, il dispositivo ad esso associato è passato da off a on.
c. Se il bit n nel registro H è a 1, il dispositivo ad esso associato è passato da on a off.
d. Se il bit n nel registro L è a 0, il dispositivo ad esso associato o non ha cambiato stato o e passato da on a off.
e. Se il bit n nel registro H è a 0, il dispositivo ad esso associato o non ha cambiato stato o è passato da off a on.

Per riassumere, con l'aiuto delle istruzioni logiche, potete rispondere alle seguenti domande tramite dei programmi eseguiti dal microcomputer:

- Lo stato logico del bit di stato è 0 o 1?
- In confronto allo stato logico precedente, il bit di stato è cambiato o è rimasto invariato?
- Se lo stato logico del bit di stato è cambiato, il cambiamento è avvenuto passando da 0 a 1 o da 1 a 0?

Passo 3

Prima di eseguire il programma, effettuate le seguenti operazioni logiche.

Sia: 10001000 = Byte di stato precedente
00001001 = Byte di stato attuale

a. 10001000 XOR 00001001

10001000 = Byte di stato precedente
00001001 = Byte di stato attuale

Il risultato di questa operazione logica vi dice quali dispositivi hanno cambiato stato. Il dispositivo n ha cambiato stato se e solo se il bit n è posto a 1 nel risultato dell'operazione XOR.

b. 10000001 AND 10001000

10001000 = Byte di stato precedente
10000001 = Risultato del punto (a), che vi dice quali dispositivi hanno cambiato stato

Il risultato di questa operazione logica vi dice quali dispositivi hanno cambiato stato passando da on a off. Il dispositivo n ha cambiato stato da on a off se e solo se il bit n è posto a 1.

c. CPL 10000000, il complemento di 10000000

10000000 è il risultato dal punto (b) e vi dice quali dispositivi hanno cambiato stato da on a off. Quindi il risultato di questa operazione logica vi dirà quali dispositivi *non* hanno cambiato stato da on a off.

d. 01111111 AND 10000001

01111111 = Risultato di (c)
10000001 = Risultato di (a)

Il risultato di questa operazione logica vi dice quali dispositivi hanno cambiato stato da off a on. Il dispositivo n ha cambiato stato da off a on se e solo se il bit n si trova posto a 1 nel risultato di questa operazione AND.

Passo 4

Eseguite il programma con il vostro microcomputer. Quali informazioni compaiono nel registro L dopo l'esecuzione?

Quei dispositivi, che hanno cambiato stato da off a on, hanno i bit ad essi associati, posti ad 1 nel registro L.

Passo 5

Quali informazioni compaiono nel registro H?

Quei dispositivi, che hanno cambiato da on a off, hanno i bit ad essi associati, posti ad 1 nel registro H.

Passo 6

Quali informazioni compaiono nel registro C?

Il byte di stato attuale, cioè quali dispositivi sono on e quali dispositivi sono off.

DOMANDE RIEPILOGATIVE

Le domande seguenti vi aiuteranno a riepilogare l'uso delle istruzioni logiche, dell'algebra Booleana, e delle operazioni logiche a più bit.

1. Eseguite queste operazioni logiche Booleane a più bit.
   a. 11001011 · 01011010
   b. 00100000 + 11011111
   c. 00100000 · 11011111
   d. 10101010 ⊕ 10100100
   e. CC · 0B
   f. A6 ⊕ 80
   g. 37 ⊕ 04
   h. 49 · 1B

2. Un byte di stato a otto bit è associato ad otto diversi dispositivi
   Bit 0: Dispositivo misuratore di pressione
   Bit 1: Dispositivo misuratore di temperatura
   Bit 2: Dispositivo misuratore di velocità
   Bit 3: Dispositivo misuratore di portata
   Bit 4: Dispositivo misuratore di tensione
   Bit 5: Dispositivo misuratore di corrente
   Bit 6: Dispositivo misuratore di livello di liquidi
   Bit 7: Dispositivo misuratore di frequenza

Per ognuno dei byte di stato esadecimali indicati qui sotto, dite quali di questi dispositivi sono on e quali sono off. Ogni bit a 1 indica che il dispositivo è on.

   a. 53
   b. 40
   c. 64
   d. 20
   e. 02
   f. 30
   g. 06
   h. C0
   i. 01
   j. 28

3. Per ogni byte di stato esadecimale indicato qui sotto (byte di stato precedente e attuale), usate le tecniche dell'algebra Booleana per determinare quali degli otto diversi dispositivi elencati nella domanda n. 2 ha cambiato stato da on a off o da off a on. Scrivete anche i calcoli di algebra Booleana che fate.

	Byte di stato precedente	Byte di stato attuale
a.	84	46
b.	2F	63
c.	02	07
d.	A7	DB

RISPOSTE

1. a. 01001010
   b. 11111111
   c. 00000000
   d. 00001110
   e. 00001000 = 08
   f. 00100110 = 26
   g. 00111111 = 3F
   h. 00001001 = 09
2. a. Dispositivi misuratori di livello di liquidi, tensione, temperatura e pressione
   b. Dispositivo misuratore di livello di liquidi
   c. Dispositivi misuratori di livello di liquidi, corrente e velocità
   d. Dispositivo misuratore di corrente
   e. Dispositivo misuratore di temperatura
   f. Dispositivi misuratori di corrente e tensione
   g. Dispositivi misuratori di velocità e temperatura
   h. Dispositivi misuratori di frequenza e livello di liquidi
   i. Dispositivo misuratore di pressione
   j. Dispositivi misuratori di corrente e portata
3. a. Convertite prima i due byte in codice binario:

   84 = 10000100 (byte di stato precedente)
   46 = 01000110 (byte di stato attuale)

   Poi, eseguite un'operazione di OR Esclusivo su questi due byte:

   $10000100 \oplus 01000110 = 11000010$

   Usando tale risultato, 11000010, eseguite un'operazione di AND fra quest'ultimo e il byte di stato precedente:

   $10000100 \cdot 11000010 = 10000000$

   Eseguite un'operazione di NOT sul risultato:

   $\overline{10000000} = 01111111$

Infine, usate il risultato dell'operazione NOT ed eseguite un'operazione AND fra il risultato stesso e il risultato dell'operazione iniziale di OR Esclusivo:

$01111111 \cdot 11000010 = 01000010$

Ora possiamo trarre le conclusioni.

1. I dispositivi misuratori di frequenza, livello di liquidi e temperatura hanno cambiato stato.
2. Il dispositivo misuratore di frequenza è passato da on a off.
3. I dispositivi misuratori di livello di liquidi e temperatura sono passati da off a on.

Guardando i due byte di stato, possiamo concludere dicendo che l'algebra Booleana ci ha fornito le risposte esatte.

b. Convertite i byte di stato esadecimali in codice binario:

2F=00101111 (byte di stato precedente)
63=01100011 (byte di stato attuale)

Eseguite un'operazione di OR Esclusivo:

$00101111 \oplus 01100011 = 01001100$

Usate il risultato facendo l'AND col byte di stato precedente:

$00101111 \cdot 01001100 = 00001100$

Finora, possiamo concludere che i dispositivi misuratori di livello di liquidi, flusso e velocità hanno cambiato stato; inoltre i dispositivi misuratori di flusso e velocità sono passati da on a off.
Complementate il risultato dell'operazione AND:

$\overline{00001100} = 11110011$

Mettete in AND questo risultato con il risultato dell'operazione XOR iniziale,

11110011 · 01001100 = 01000000

Perciò, il dispositivo misuratore di livello di liquidi è passato da off a on.

c. Convertite i byte di stato esadecimali in codice binario,

02 = 00000010 (byte di stato precedente)
07 = 00000111 (byte di stato attuale)

Eseguite un'operazione di XOR:

00000010 ⊕ 00000111 = 00000101

Usate questo risultato per eseguire un'operazione di AND con il byte di stato precedente:

00000010 · 00000101 = 00000000

I dispositivi misuratori di velocità e di pressione hanno cambiato stato. Nessuno dei due è passato da on a off.
Complementate il risultato dell'operazione AND:

$\overline{00000000} = 11111111$

Mettete in AND questo risultato con il risultato dell'operazione di XOR iniziale:

11111111 · 00000101 = 00000101

Sia il dispositivo misuratore di velocità che quello misuratore di pressione sono passati da off a on.

d. Convertite i byte di stato esadecimali in codice binario:

A7 = 10100111 (byte di stato precedente)
DB = 11011011 (byte di stato attuale)

Eseguite un'operazione di XOR:

10100111 ⊕ 11011011 = 01111100

Usate questo risultato per eseguire un'operazione di AND con il byte di stato precedente:

01111100 · 10100111 = 00100100

I dispositivi misuratori di livello di liquidi, di corrente, di velocità, di portata e di tensione hanno cambiato stato. I dispositivi misuratori di corrente e di velocità sono passati da on a off.
Complementate il risultato dell'operazione di AND:

$\overline{00100100} = 11011011$

Mettete in AND questo risultato con il risultato dell'operazione di XOR iniziale:

11011011 · 01111100 = 01011000

I dispositivi misuratori di livello di liquidi, di tensione e di flusso sono passati da off a on.

CAPITOLO 10

MANIPOLAZIONE DEI BIT, ISTRUZIONI DI ROTAZIONE E SHIFT

INTRODUZIONE

In questo capitolo prenderemo in considerazione due gruppi di istruzioni particolarmente significative del Set di istruzioni dello Z-80, il gruppo di *manipolazione dei bit* e il *gruppo di rotazione e shift*. Le istruzioni di manipolazione dei bit vi permetteranno di controllare e/o modificare i valori di tutti i registri e di tutte le locazioni di memoria a livello dei singoli bit; in effetti, le istruzioni di manipolazione dei bit costituiscono più del 50% delle nuove istruzioni dello Z-80, non disponibili sui sistemi 8080.

OBIETTIVI

Al termine di questo capitolo, sarete in grado di:

- Usare le istruzioni di manipolazione dei bit, BIT, SET e RESET
- Usare le istruzioni di rotazione e di shift
- Capire l'utilità di ognuna delle istruzioni di manipolazione dei bit, di rotazione e di shift
- Capire come si applicano le istruzioni di RRD e RLD per elaborare i numeri BCD.

	BIT	TRA REGISTRI							INDIR. TRAM. REG.	INDICIZZ.	
		A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
TEST DI UN "BIT"	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
AZZERAMENTO DI UN BIT "RES"	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET DI UN BIT "SET"	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

Tabella 10-1. Gruppo manipolazione di bit.

OPERAZIONI DI SET, RESET E TEST DI BIT

Un esempio semplice dell'uso delle operazioni SET, RESET e TEST BIT è dato dalla procedura, seguita in America dai postini rurali e dai residenti, per facilitare il ritiro e la consegna della posta. Su ogni cassetta postale situata su una strada di campagna è presente una bandierina rossa che può essere girata verso l'alto, flag settato (stato logico uno), o verso il basso, flag resettato (stato logico zero).



Posizione dei flags.

Se vuole che il postino si fermi davanti alla cassetta per ritirare delle lettere, il residente *setta* (SET) il flag, solleva cioé la bandierina verso l'alto. Il postino, passando, guarda (TEST del flag) per vedere se la bandierina è alzata. In caso affermativo postino si ferma, preleva le lettere e poi riabbassa la bandierina (RESET) in modo che la cosa può ripetersi secondo la stessa convenzione il giorno seguente.
Se la bandierina non è alzata, il postino non "serve" la cassetta postale, a meno che non abbia delle lettere da consegnare al residente. Perciò senza bandierina, l'unica volta in cui il residente potrebbe inviare una lettera sarebbe in quei giorni in cui egli stesso riceve delle lettere, per cui il postino deve fermarsi in ogni caso. Questa sarebbe una situazione scomoda per coloro che non ricevono mai posta.

Con questo tipo di convenzione, basata su di una bandierina, parecchie informazioni convergono dal residente al postino.
Questa procedura funziona perchè c'è un accordo tra il residente e il postino per quanto riguarda le condizioni nelle quali ognuno di loro *alzerà o abbasserà* la bandierina. Questo è un esempio di semplicissimo protocollo. Secondo il Webster, un *protocollo* può essere definito come un codice che prescrive uno stretto rispetto di norme formali e di precedenze. Il concetto di protocollo sarà per voi molto importante quando inizierete a progettare interfacce fra i microcomputer e altri dispositivi, o fra 2 microcomputer.

Il set di istruzioni dello Z-80 comprende 240 diverse istruzioni di manipolazione di bit

80 istruzioni di set bit ('SET')
80 istruzioni di reset bit ('RES')
80 istruzioni di test bit ('BIT')

Per esempio, l'istruzione di set bit, SET 0,A (codice operativo: CB C7) inserisce un 1 nel bit 0 dell'accumulatore.

Vediamo questi due esempi:

	Accumulatore
Esempio 1	
Prima dell'esecuzione di SET 0,A	1 1 0 0 1 0 0 0
Dopo l'esecuzione di SET 0,A	1 1 0 0 1 0 0 1
Esempio 2	
Prima dell'esecuzione di SET 0,A	1 1 1 1 0 0 1 1
Dopo l'esecuzione di SET 0,A	1 1 1 1 0 0 1 1

Ricordate che gli otto bit di qualunque registro o locazione di memoria sono sempre numerati da destra a sinistra iniziando con il bit zero e finendo con il bit sette.
Un modo comune per descrivere l'istruzione di SET 0,A è dire che essa *"setta"* il bit zero dell'accumulatore 0 ad 1.
Un modo più breve ed anche più comune per indicare la stessa cosa è dire che l'istruzione *"setta"* il bit zero dell'accumulatore. L'uso comune ha determinato che l'espressione *"settare* un bit" significhi che il valore del bit sia 1 dopo che l'istruzione è stata eseguita. Analogamente, la frase *"resettare* un bit" significa che il valore del bit è 0 dopo che è stata eseguita l'istruzione di RESET.
Dovreste notare che sia l'istruzione di set bit che quella di reset bit non fanno riferimento al valore originale del bit. Quindi un bit precedentemente settato può essere settato ancora e in tale sequenza non verrà modificato.
L'istruzione di RES 0,D (codice operativo: CB 82) resetterà il bit zero del registro D.

	Registro D
Esempio 3	
Prima dell'esecuzione di RES 0,D	1 1 0 0 1 1 1 1
Dopo l'esecuzione di RES 0,D	1 1 0 0 1 1 1 0
Esempio 4	
Prima dell'esecuzione di RES 0,D	0 0 0 0 0 0 0 0
Dopo l'esecuzione di RES 0,D	0 0 0 0 0 0 0 0

Nè l'istruzione di SET nè quella di RESET alterano alcun flag.

Un'istruzione un po' complicata è quella di test bit. Per esempio l'istruzione BIT 0,A (codice operativo: CB 47) effettua un test sul bit zero dell'accumulatore. Se il bit zero dell'accumulatore è zero, allora questa istruzione setterà il flag di zero. Ovvero, il valore del flag di zero sarà 1 se il bit zero dell'accumulatore contiene zero.

	Accumulatore	Flag di zero
Esempio 5		
Prima dell'esecuzione di BIT 0,A	1 0 0 1 1 1 0 0	X
Dopo l'esecuzione di BIT 0,A	1 0 0 1 1 1 0 0	1
Esempio 6		
Prima dell'esecuzione di BIT 0,A	1 1 1 0 0 1 1 1	X
Dopo l'esecuzione di BIT 0,A	1 1 1 0 0 1 1 1	0
Esempio 7		
Prima dell'esecuzione di BIT 0,A	1 1 1 1 1 1 1 1	X
Dopo l'esecuzione di BIT 0,A	1 1 1 1 1 1 1 1	0

(X = non ha importanza)

Notate che questa istruzione, BIT 0,A in nessun caso modifica il valore dell'accumulatore. Troverete che l'istruzione di test bit è molto utile. Per esempio, questa istruzione permette di determinare il valore di un particolare bit, in qualunque registro o locazione di memoria senza dover creare un byte di maschera e senza alterare il contenuto dell'accumulatore.

Prendiamo in considerazione queste due sequenze di istruzioni che attuano la stessa funzione, ovvero quella di determinare se il bit D4 della locazione 01FF è a 0:

Sequenza 1: LD A,(01FFH).
AND 08H

Sequenza 2: LD A,(01FFH)
BIT 4,A

La sequenza 1 usa il byte di maschera 08, per cambiare il contenuto dell'accumulatore, con il flag di zero che conterrà 0 o 1 a seconda del risultato. la sequenza 2 influenza I flag di zero esattamente nello stesso modo della sequenza 1, senza però cambiare il contenuto del registro A. Entrambe le sequenze richiedono 5 byte di memoria.

Sorgente e Destinazione

Tipo di Rotazione o Scorrimento	A	B	C	D	E	H	L	(HL)	(IX + d)	(IY + d)
'RLC'	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
'RRC'	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
'RL'	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
'RR'	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
'SLA'	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26.	DD CB d 26	FD CB d 26
'SRA'	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
'SRL'	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
'RLD'								ED 6F		
'RRD'								ED 67		

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F

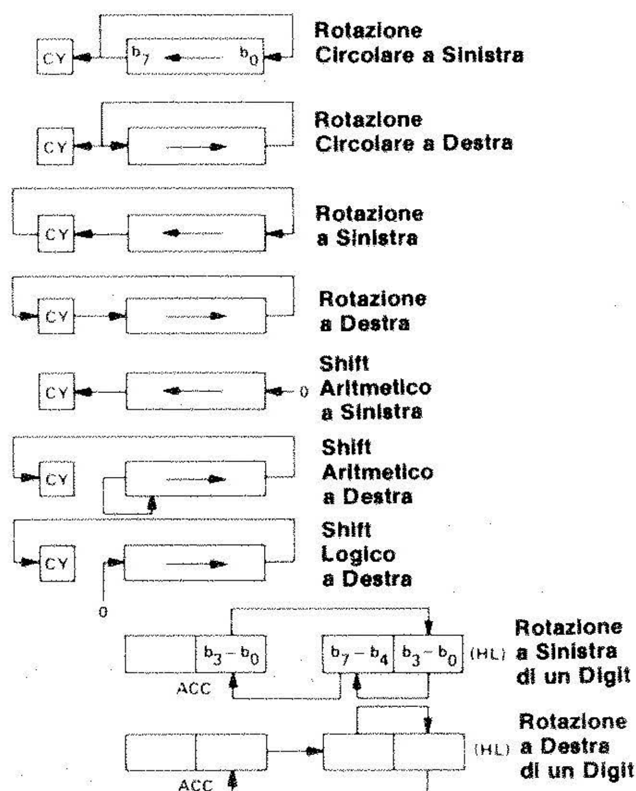


Tabella 10-2. Rotazioni e scorrimenti.

GRUPPO DI ISTRUZIONI DI ROTAZIONE E DI SHIFT

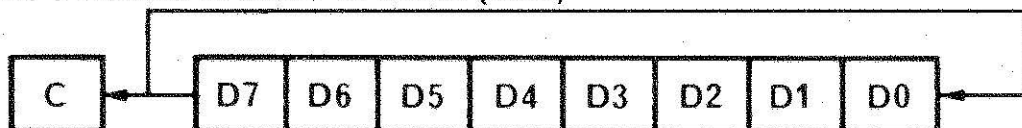
Per lo Z-80, ci sono 74 istruzioni di questo tipo. All'interno del tale gruppo le 4 istruzioni dell'INTEL 8080A di rotazione dell'accumulatore (RLCA, RRCA, RLA e RRA) sono di fatto abbastanza ridondanti e sono state incluse nel set di istruzioni dello Z-80 solo per mantenere una completa compatibilità con l'8080. Dovreste notare che queste quattro istruzioni influenzano solo il flag di carry. L'esecuzione di una qualunque di queste quattro istruzioni non influenza invece il flag di Zero, di parità/overflow o di segno. Le 70 istruzioni rimanenti influenzano invece tali flag.

In aggiunta a queste 74 istruzioni, ci sono due particolari istruzioni di rotazione per digit decimali, di cui parleremo separatamente.

ISTRUZIONI DI ROTAZIONE

Consideriamo prima le quattro classi di istruzioni di rotazione.
Per ogni istruzione vi diamo parecchi diagrammi che illustrano la particolare operazione di rotazione da essa effettuata.

Rotazione a sinistra in modo circolare (RLC)



Possiamo interpretare questo diagramma esaminando il contenuto del bit di carry e dell'accumulatore sia prima che dopo l'esecuzione dell'istruzione di RLC A.

	Bit di Carry	Accumulatore
Prima dell'esecuzione di RLC A	C	D7 D6 D5 D4 D3 D2 D1 D0
Dopo l'esecuzione di RLC A	D7	D6 D5 D4 D3 D2 D1 D0 D7

Per esempio:

Prima dell'esecuzione di RLC A	X	1 1 0 1 1 1 0 0
Dopo l'esecuzione di RLC A	1	1 0 1 1 1 0 0 1
(X= non significativo)		

Rotazione a destra in modo circolare (RRC)



Da questo diagramma si può ricavare che vengono apportati i seguenti cambiamenti al bit di carry e al registro C durante l'esecuzione della istruzione di RRC C:

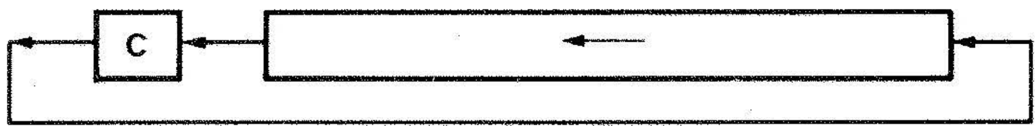
	Bit di Carry	Registro C
Prima dell'esecuzione di RRC C	C	D7 D6 D5 D4 D3 D2 D1 D0
Dopo l'esecuzione di RRC C	D0	D0 D7 D6 D5 D4 D3 D2 D1

Per esempio:

Prima dell'esecuzione di RRC C	X	1 1 0 1 1 1 0 0
Dopo l'esecuzione di RRC C	0	0 1 1 0 1 1 1 0

Notate che, sia per l'istruzione RRL che per l'istruzione RRC, il contenuto originario del flag di carry è distrutto dall'esecuzione dell'istruzione.

Rotazione verso sinistra (RL)



Questo diagramma indica che vengono apportati i seguenti cambiamenti al bit di carry e all'accumulatore nel caso dell'esecuzione dell'istruzione RL A:

	Bit di Carry	Accumulatore
Prima dell'esecuzione di RL A	C	D7 D6 D5 D4 D3 D2 D1 D0
Dopo l'esecuzione di RL A	D7	D6 D5 D4 D3 D2 D1 D0 C

Per esempio:

Prima dell'esecuzione di RL A	0	1 1 0 1 1 1 0 0
Dopo l'esecuzione di RL A	1	1 0 1 1 1 0 0 0

Rotazione verso destra (RR)



Da questo diagramma si può ricavare che vengono apportati i seguenti cambiamenti al bit di carry e al registro D nel corso dell'istruzione RR D:

	Bit di Carry	Registro D
Prima dell'esecuzione di RR D	C	D7 D6 D5 D4 D3 D2 D1 D0
Dopo l'esecuzione di RR D	D0	C D7 D6 D5 D4 D3 D2 D1

Per esempio:

Prima dell'esecuzione di RR D	0	1 1 0 1 1 1 0 0
Dopo l'esecuzione di RR D	0	0 1 1 0 1 1 1 0

Le quattro istruzioni di rotazione vengono usate spesso per esaminare uno dopo l'altro i bit presenti in un registro o in una locazione di memoria particolari.
Per esempio, se volete trovare il bit diverso da zero di ordine più alto nell'accumulatore, potete eseguire successive istruzioni di rotazione verso sinistra finchè si trova che il flag di carry è settato a 1:

```
        LD A,X          ;Carica l'accumulatore con il byte X che deve essere
                        ;testato
        LD C,08H        ;Contatore dei bit
CHECK:  DEC C           ;Aggiorna il contatore dei bit
        JP M,END        ;Tutti i byte sono stati controllati?
        RL A            ;Ruota il prossimo bit più significativo nel Carry
        JP NC,CHECK     ;E' 0 o 1? se è 0, passa al bit successivo
END:    RST 38H         ;Altrimenti restituisci il controllo al sistema
                        ;operativo.
```

Nella sequenza di istruzioni che avete visto, il registro C contiene il numero (7 nel caso del bit più significativo ......, 0 per il bit meno significativo) che indica la posizione del bit non-zero di ordine più alto nell'accumulatore. Notate che potrebbe essere usata l'istruzione ad un byte RLA al posto della istruzione a due byte RL A, perchè in questo caso solo il flag di carry è importante per la logica del programma.
Nell'esempio che segue potete vedere un'altra applicazione interessante delle istruzioni di rotazione. Supponiamo che debbano essere eseguiti uno dopo l'altro un certo numero di processi, cioè prima 1, poi 2, ecc.. Tuttavia non tutti i processi vengono sempre eseguiti; per esempio, in alcune situazioni la sequenza appropriata può essere la seguente:

Processo 1 Processo 3 Processo 5 Processo 8	oppure	Processo 5 Processo 7	oppure	Processo 3 Processo 6 Processo 8

o una qualunque delle 256 sequenze possibili.

Per implementare il programma richiesto per trattare questa situazione, si può usare una istruzione di rotazione verso destra. Per prima cosa, vengono sviluppate delle subroutine per ogni processo, SUB 1 per il processo 1 ....., SUB 8 per il processo 8. Quindi, ogni volta che deve essere eseguita una sequenza di processi viene generata su di un byte una descrizione della sequenza: il bit 0 viene settato se deve essere eseguito il processo 1, viene settato il bit 1 per il processo numero 2, ..., viene settato il bit 7 per il processo 8. Se un bit è a 0, il processo ad esso associato non viene gestito. La logica di salto per realizzare una sequenza di questo tipo può essere la seguente: (notate che l'istruzione di RRA è quasi identica all'istruzione di RR)

```
RRA
CALL C,SUB1
RRA
CALL C,SUB2
RRA
CALL C,SUB3
RRA
CALL C,SUB4
RRA
CALL C,SUB5
RRA
CALL C,SUB6
RRA
CALL C,SUB7
RRA
CALL C,SUB8
```

dove supponiamo che il byte di descrizione della sequenza sia caricato nell'accumulatore. Notate che tale sequenza di istruzioni usa l'istruzione ad un byte RRA invece dell'istruzione a due byte RR A.

Il prossimo gruppo di istruzioni di cui parleremo si chiama GRUPPO DI SHIFT (scorrimento). Le istruzioni di rotazione e di Shift vengono spesso usate insieme per eseguire molte importanti funzioni di programmazione. Vi daremo a tal proposito molti esempi in questo capitolo e in quelli successivi.

ISTRUZIONI DI SHIFT

Shift aritmetico verso sinistra (SLA)



Da questo diagramma si può ricavare che vengono apportati i seguenti cambiamenti al bit di carry e all'accumulatore:

	Bit di Carry	Accumulatore
Prima dell'esecuzione di SLA A	C	D7 D6 D5 D4 D3 D2 D1 D0
Dopo l'esecuzione di SLA A	D7	D6 D5 D4 D3 D2 D1 D0 0

Questa istruzione ha l'effetto di moltiplicare il contenuto dell'accumulatore per 2, dando una segnalazione di overflow quando il bit di carry è posto a 1. Vediamo alcuni esempi:

Esempio 1

	Bit di Carry	Accumulatore
Prima dell'esecuzione di SLA A	X	0 0 1 1 0 0 1 1 = 51 (decim.)
Dopo l'esecuzione di SLA A	0 (non overflow)	0 1 1 0 0 1 1 0 = 102 (decim.)

Esempio 2

Prima dell'esecuzione di SLA A	X	0 1 1 0 0 1 1 0 = 102 (decim.)
Dopo l'esecuzione di SLA A	0(non overflow)	1 1 0 0 1 1 0 0 = 204 (decim.)

Esempio 3

Prima dell'esecuzione di SLA A	X	1 0 0 1 1 0 0 0 = 152 (decim.)
Dopo l'esecuzione di SLA A	1(overflow)	0 0 1 1 0 0 0 0 = 48 (decim.)

Nei primi due esempi il contenuto dell'accumulatore è stato moltiplicato per due dall'istruzione a SLA A. Ma nell'esempio 3 il contenuto dell'accumulatore invece che raddoppiare è andato da 152 a 48. La ragione di questo sta nel fatto che 152 moltiplicato per due dà 304, maggiore di 256, l'intero positivo più grande che possa essere rappresentato con 8 bit (usando la rappresentazione binaria, non quella complemento a due). Quando il numero originario che viene shiftato è maggiore di 128, il risultato dello shift non sarà il numero moltiplicato per due, ma il doppio del numero meno 256 e per indicare questo verrà settato il bit di carry.

L'istruzione SLA può anche essere usata insieme all'istruzione RL, per eseguire shift verso sinistra di più byte. La sequenza di istruzioni che segue esegue uno shift di un bit verso sinistra della coppia di registri DE, riempiendo con uno zero il bit lasciato vuoto:

```
SLA E
RL D
```

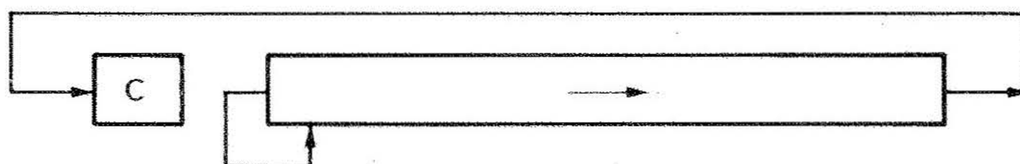
Notate che possiamo effettuare uno shift verso sinistra anche di coppie di byte di

memoria. Per effettuare uno shift a sinistra, riempiendo con uno zero la posizione libera, della coppia di locazioni di memoria 0100 e 0101, possiamo usare la seguente sequenza di istruzioni:

```
LD IX,0100H
SLA (IX)
RL (IX+1H)
```

Gli shift a più byte sono estremamente importanti nelle operazioni aritmetiche su più byte, come le moltiplicazioni e le divisioni.

Shift aritmetico verso destra (SRA)



Da questo diagramma si può ricavare che sono effettuati i seguenti cambiamenti al bit di carry e all'accumulatore:

	Bit di Carry	Accumulatore
Prima dell'esecuzione di SRA A	C	D7 D6 D5 D4 D3 D2 D1 D0
Dopo l'esecuzione di SRA A	D0	D7 D7 D6 D5 D4 D3 D2 D1

Notate che il valore del bit di carry viene alterato.
Questa istruzione ha l'effetto di dividere il contenuto di un certo registro per 2, mettendo il resto di questa divisione nel bit di carry.
La divisione viene eseguita nella forma del complemento a due, cioè l'istruzione suppone che il numero contenuto nel registro sia nella forma del complemento a due e genera un quoziente, posto nello stesso registro, e nella stessa rappresentazione. Vediamo i seguenti esempi:

	Bit di Carry	Accumulatore
Esempio 4		
Prima dell'esecuzione di SRA A	X	0 0 0 0 1 1 1 1 = 15 (decimale)
Dopo l'esecuzione di SRA A	1 (resto)	0 0 0 0 0 1 1 1 = 7 (decimale)
Esempio 5		
Prima dell'esecuzione di SRA	X	1 0 0 0 1 1 1 0 =-114 (decimale)
Dopo l'esecuzione di SRA A	0 (resto)	1 1 0 0 0 1 1 1 =- 57 (decimale)

L'istruzione SRA può essere usata insieme all'istruzione di RR per eseguire shift a destra su dati più byte, nello stesso modo in cui le istruzioni di SLA e RL eseguono shift a sinistra. Per esempio, la coppia dei registri HL può essere shiftata a destra, il bit D7 del registro H essendo lasciato inalterato, come segue:

```
SRA H
RR L
```

Shift logico verso destra (SRL)



	Bit di Carry	Accumulatore
Prima dell'esecuzione di SRL A	C	D7 D6 D5 D4 D3 D2 D1 D0
Dopo l'esecuzione di SRL A	D0	0 D7 D6 D5 D4 D3 D2 D1

Questa istruzione ha l'effetto di dividere il numero nel registro per 2, il numero contenuto nel registro essendo visto come un numero binario positivo a 8 bit. Dopo l'esecuzione di tale istruzione il quoziente è presente nello stesso registro ed il resto nel bit di carry.

Esempio 6

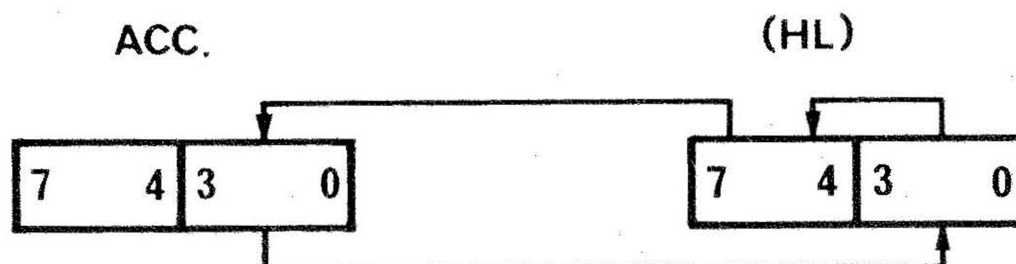
	Bit di Carry	Accumulatore
Prima dell' esecuzione di SRL A	X	1 0 0 0 0 0 1 1 = 131 (decimale)
Dopo l'esecuzione di SRL A	1 (resto)	0 1 0 0 0 0 0 1 = 65 (decimale)

Per shiftare a destra dei dati a più byte, potete usare le istruzioni di SRL e RR; per esempio, per lo shift di una sequenza di 8 byte che risiede nelle locazioni di memoria da 0100 a 0107 potete usare le istruzioni:

```
        LD B,08H
        LD HL,0107H
        SRL (HL)
SHIFT:  DEC HL
        DEC B
        JP Z,END
        RR (HL)
        JP SHIFT
END:    RST 38H
```

Parliamo ora delle due particolari istruzioni di rotazione di cifre (digit) decimali. Descriviamo prima le istruzioni di RLD e RRD e vi facciamo poi degli esempi del loro uso.

Rotazione a sinistra di un digit (RLD)



	Accumulatore								Cella di memoria puntata da HL							
Prima dell' esecuzione di RLD	D7	D6	D5	D4	D3	D2	D1	D0	B7	B6	B5	B4	B3	B2	B1	B0
Dopo l'esecuzione di RLD	D7	D6	D5	D4	B7	B6	B5	B4	B3	B2	B1	B0	D3	D2	D1	D0

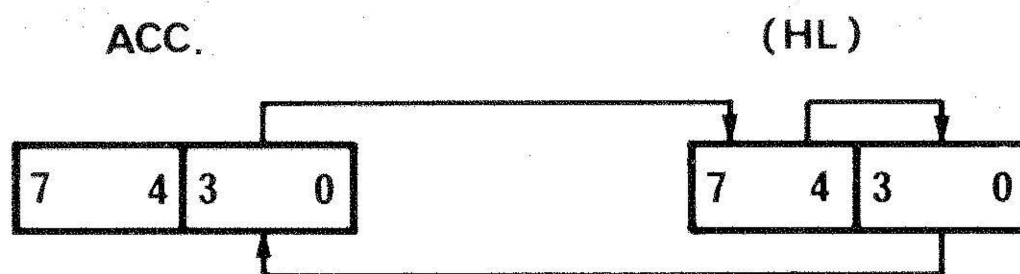
Per esempio:

Prima dell'esecuzione di RLD	1	1	0	1	1	0	0	0	0	0	1	0	1	1	0	0
Dopo l'esecuzione di RLD	1	1	0	1	0	0	1	0	1	1	0	0	1	0	0	0

Notate che questa istruzione fa ruotare metà dei byte (metà byte, cioè quattro bit, è anche detto nibble) per volta. Non fa cioè ruotare i singoli bit, ma i nibble presi come elementi non divisibili.

In altre parole, i 4 bit 1000 nel nibble meno significativo dell'accumulatore compaiono nella loro destinazione finale come 1000, non come 0001.

Rotazione a destra di un digit (RRD)



	Accumulatore								Cella di memoria puntata da HL							
Prima dell'esecuzione di RRD	D7	D6	D5	D4	D3	D2	D1	D0	B7	B6	B5	B4	B3	B2	B1	B0
Dopo l'esecuzione di RRD	D7	D6	D5	D4	B3	B2	B1	B0	D3	D2	D1	D0	B7	B6	B5	B4

Per esempio:

Prima dell'esecuzione di RRD	1	1	0	1	0	0	0	1	1	0	1	1	1	1	1	0
Dopo l'esecuzione di RRD	1	1	0	1	1	1	1	0	0	0	0	1	1	0	1	1

Le istruzioni RRD e RLD sono particolarmente utili quando devono essere trattati dei nibble (cioè, di fatto, dei digit) anzichè dei bit o dei byte. I programmi che usano, per i numeri, la rappresentazione decimale codificata binaria (BCD), sono ottimi esempi di questo tipo di elaborazione orientata verso il digit. Ricordate che la rappresentazione dei numeri BCD utilizza quattro bit per ogni cifra decimale, permettendo così di rappresentare due cifre decimali per ogni byte. Per esempio la rappresentazione BCD del numero decimale 83 è

1 0 0 0	0 0 1 1
prima cifra = 8	seconda cifra = 3

Viceversa, l'interpretazione BCD del byte

<u>0 1 1 1</u> <u>0 0 0 0</u>

porta ad ottenere il numero decimale 70. Quindi, per quanto riguarda la rappresentazione BCD, l'istruzione RLD fa ruotare i digit decimali verso sinistra fra la cella di memoria puntata da HL e l'accumulatore. Analogamente, l'istruzione RRD fa ruotare i digit BCD verso destra.
Per esempio la seguente sequenza di istruzioni shifterà verso sinistra un gruppo di otto digit BCD che risiedono nelle quattro locazioni di memoria contigue da 0100 a 0103:

```
         LD B,04H
         LD HL,0100H
         LD A,00H
ANCORA:  RLD
         INC HL
         DEC B
         JP NZ,ANCORA
         RST 38H
```

Notate che il digit decimale più significativo (il digit di ordine alto della locazione di memoria 0103) viene trasferito nel nibble di ordine basso dell'accumulatore.

Il digit alto dell'accumulatore non viene mai alterato da queste istruzioni. Il diagramma seguente illustra gli effetti della suddetta routine sull'accumulatore e sulle locazioni di memoria da 0100 a 0103.

	Accumulatore	0103	0102	0101	0100
Prima:	X X	8 7	6 5	4 3	2 1
Dopo:	0 8	7 6	5 4	3 2	1 0

Nell'Esperimento N. 2 viene rappresentata un'altra applicazione di queste istruzioni di rotazione, che vengono utilizzate per effettuare una traduzione dalla rappresentazione BCD e quella ASCII.

INTRODUZIONE AGLI ESPERIMENTI

Gli esperimenti che seguono hanno lo scopo di aiutarvi a capire come funzionano le istruzioni di manipolazione dei bit, di rotazione e di shift, e di illustrarvene l'uso. Nell'Esperimento N. 1 vi diamo la rappresentazione ASCII dei numeri. Nel corso dei tre esperimenti presentiamo dei programmi che effettuano la conversione dei numeri tra le diverse rappresentazioni, binaria, BCD e ASCII.

Esperimento N.	Commenti
1	Illustra l'uso delle istruzioni BIT e RR in programmi di conversione dalla rappresentazione binaria a quella ASCII.
2	Illustra l'uso dell'istruzione di RL in un programma di conversione dalla rappresentazione ASCII a quella binaria.
3	Illustra l'uso dell'istruzione di RRD in un programma di conversione dalla rappresentazione BCD a quella ASCII.

ESPERIMENTO N. 1

Scopo

Questo esperimento illustra l'uso dell'istruzione di BIT in un programma di conversione dalla rappresentazione binaria a quella ASCII. La rappresentazione ASCII dei numeri da 0 a 9 è riportata nella seguente tabella:

Numeri decimali	Rappresentazione ASCII in codice esadecimale
0	30
1	31
2	32
3	33
4	34
5	35
6	36
7	37
8	38
9	39

L'ASCII è semplicemente un'altro metodo per rappresentare i numeri usando per ogni numero un byte a otto bit. Per molti CRT, terminali e stampanti, ASCII è il codice standard per la rappresentazione di numeri, lettere e caratteri speciali come ; , . , ! , ?, ecc.

Convertire da binario in ASCII significa prendere una stringa (cioè una sequenza) di 0 e di 1 e ricavare una stringa corrispondente formata da 30 e da 31.
Per esempio il byte 0 1 0 0 1 1 0 1 verrebbe convertito nella seguente sequenza di byte (rappresentata in codice esadecimale):

30 31 30 30 31 31 30 31

Il programma seguente converte il byte, che si trova attualmente nel registro B, in una stringa di otto byte che viene memorizzata a partire dalla locazione di memoria 0200.

Programma N. 29

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
0100	0E 08		LD C,08H	;Contatore dei bit
0102	21 00 02		LD HL,0200H	;Inizio della stringa ASCII
0105	36 30	NXTBIT:	LD (HL),30H	;Zero in ASCII
0107	CB 40		BIT 0,B	;Controlla il bit
0109	28 01		JR Z,ZERO	;Se non è zero, incrementa ;da 30 a 31
010B	34		INC (HL)	
010C	23	ZERO:	INC HL	;Incrementa il puntatore della stringa ;ASCII
010D	CB 18		RR B	;Fai ruotare B per accedere al bit ;seguente
010F	0D		DEC C	;Aggiorna il contatore dei bit
0110	20 F3		JR NZ,NXTBIT	;Sono stati convertiti tutti i bit?
0112	FF		RST 38H	

Passo 1

Caricate il programma all'indirizzo indicato. Eseguitelo usando parecchi byte diversi, caricandoli nel registro B. Per esempio se il contenuto del registro B è

0 1 1 1 1 0 0 0

allora le locazioni di memoria da 0200 a 0207 dovrebbero contenere (in codice esadecimale):

(0200)=30
(0201)=30
(0202)=30
(0203)=31
(0204)=31
(0205)=31
(0206)=31
(0207)=30

Passo 2

Tentate di scrivere il programma sopra descritto utilizzando, per la conversione da ASCII a binario, una istruzione di rotazione e il flag di carry. Mettete a punto e provate il vostro programma per essere sicuri che funzioni correttamente.

ESPERIMENTO N. 2

Scopo

Questo esperimento illustra l'uso dell'istruzione RL in un programma che effettua la conversione dalla rappresentazione ASCII a quella binaria. In questo programma, si ha come dato di partenza una stringa di otto caratteri ASCII che rappresentano 0 o 1 (30 e 31 in codice esadecimale) e viene generato come risultato un solo byte con i bit correttamenti posti a 0 o 1 logico. Il programma seguente converte una stringa di otto byte, a partire dalla locazione 0200, in un solo byte che è contenuto nel registro B.

Programma N. 30

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
0120	0E 08		LD C,08H	;Contatore di bit
0122	21 00 02		LD HL,0200H	;Indirizzo di inizio della stringa ASCII
0125	7E	NXT:	LD A,(HL)	;Prendi il byte ASCII
0126	1F		RRA	;Sposta il bit 0 nel flag C
0127	CB 18		RR B	;Fai ruotare il flag C nel registro B
0129	23		INC HL	;Punta il byte successivo
012A	0D		DEC C	;Aggiorna il contatore dei bit
012B	20 F8		JR NZ,NXT	;Abbiamo convertito tutti i byte ASCII?
012D	FF		RST 38H	

Passo 1

Caricate ed eseguite il programma, introducendo un certo numero di stringhe ASCII diverse.

Passo 2

Riscrivete il programma in modo da elaborare una stringa di byte ASCII che parta dalla locazione 0207 fino alla 0200. In che modo questo fatto influenzerà le istruzioni di rotazione? Mettete a punto e provate il vostro programma in modo da essere sicuri che funzioni correttamente.

ESPERIMENTO N. 3

Scopo

Questo esperimento illustra l'uso dell'istruzione di RRD in un programma che effettua la conversione dalla rappresentazione BCD a quella ASCII. Fare questa conversione significa prendere una stringa di byte "BCD impaccati", cioè byte che contengono due cifre BCD ed elaborarli in modo da generare una stringa di cifre ASCII, una cifra per byte. Per esempio, i valori BCD:

Byte 1	0 0 1 1	1 0 0 1	= 39 (BCD)
Byte 2	0 1 0 1	1 0 0 0	= 58 (BCD)
Byte 3	0 0 0 0	0 0 0 1	= 01 (BCD)
Byte 4	0 1 1 1	0 0 0 0	= 70 (BCD)

vengono convertiti nella stringa ASCII di otto byte (scritta in codice esadecimale)

33 39 35 38 30 31 37 30

Il programma seguente converte una stringa di byte contenenti codici BCD impaccati, il cui indirizzo d'inizio è contenuto nella coppia di registri HL, in una stringa di byte ASCII il cui indirizzo d'inizio è contenuto nella coppia di registri DE. HL è posto inizialmente a 0210 e DE a 0301. Il registro C contiene il numero di byte BCD impaccati da convertire. Abbiamo posto il contenuto di C uguale a 04.

Programma N. 31

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
0130	3E 30		LD A,30H	;Inizializza il nibble di ordine alto ;dell'accumulatore a 3
0132	21 10 02		LD HL,0210H	;Indirizzo sorgente (BCD impaccati)
0135	11 01 03		LD DE,0301H	;Indirizzo destinazione (stringa ASCII)
0138	0E 04		LD C,04H	;Numero dei byte sorgente
013A	ED 67	BCD:	BRD	;Fai ruotare il nibble di ordine basso ;nell'accumulatore. Dato che il nibble ;di ordine alto è 3, abbiamo ;nell'accumulatore il carattere ASCII ;corrispondente
013C	12		LD (DE),A	;Memorizza il byte ASCII
013D	1B		DEC DE	;Decrementa il puntatore di destinazione ;per la cifra BCD di ordine alto
013E	ED 67		RRD	;Fai ruotare il nibble di ordine alto ;nell'accumulatore
0140	12		LD (DE),A	;Memorizza il byte ASCII
0141	ED 67		RRD	;Fai ruotare di nuovo il nibble di ordine ;alto nel byte sorgente ripristinando ;il dato iniziale
0143	13		INC DE	;Aggiorna il puntatore di destinazione
0144	13		INC DE	
0145	13		INC DE	
0146	23		INC HL	;Aggiorna il puntatore sorgente
0147	0D		DEC C	;Aggiorna il contatore dei byte ;sorgente
0148	20 F0		JR NZ,BCD	;Abbiamo terminato?
014A	FF		RST 38H	

Passo 1

Caricare il programma all'indirizzo indicato. Eseguite il programma passo passo, usando stringhe diverse di numeri BCD impaccati per cercare di capire come il programma opera.

Passo 2

Parliamo ora del funzionamento del programma precedente. Prendiamo in considerazione il diagramma della figura 10-1.

Locazione di memoria	Numero BCD di ordine più alto D7 D6 D5 D4	Numero BCD di ordine più basso D3 D2 D1 D0
0210	BCD1	BCD2
0211	BCD3	BCD4
0212	BCD5	BCD6
0213	BCD7	BCD8

0300	ASCII 1	Caricato dopo il 2° RRD
0301	ASCII 2	Caricato dopo il 1° RRD
0302	ASCII 3	Caricato dopo il 5° RRD
0303	ASCII 4	Caricato dopo il 4° RRD
0304	ASCII 5	Caricato dopo l' 8° RRD
0305	ASCII 6	Caricato dopo il 7° RRD
0306	ASCII 7	Caricato dopo l' 11° RRD
0307	ASCII 8	Caricato dopo il 10° RRD

	Accumulatore		Locazione di memoria 0210	
Configurazione iniziale	3	0	BCD1	BCD2
Dopo la prima istruzione RRD	3	BCD2	0	BCD1

Nota: 3 BCD2 = ASCII2 è memorizzato in (0301)

	Accumulatore		Locazione di memoria 0210	
Dopo la seconda istruzione RRD	3	BCD1	BCD2	0

Nota: 3 BCD1 = ASCII1 è memorizzato in (0300)

	Accumulatore		Locazione di memoria 0210	
Dopo la terza istruzione RRD	3	0	BCD1	BCD2

Nota: tutto è stato ripristinato e il trasferimento del byte sorgente è completo.

Figura 10-1

Vi sono tre cose particolarmente importanti da capire riguardo questo programma:

1. L'istruzione di RRD serve a spostare i nibble fra la memoria e l'accumulatore come potete vedere nella figura 10-1.

2. Per eseguire l'istruzione RRD, è necessario che BCD2 sia convertito in ASCII2 *prima* che BCD1 sia convertito in ASCII1. Ciò spiega perché il registro DE è inizializzato a 0301, decrementato e poi incrementato tre volte.

3. L'istruzione di RRD ci permette di inizializzare l'accumulatore a 30 e di eseguire la traduzione da BCD in ASCII semplicemente facendo ruotare i digit BCD nel digit di ordine basso, mantenendo costante il nibble di ordine alto.

Notate che questo programma si stacca dalla nostra abitudine di associare alle parti più significative dei dati le locazioni di memoria più alte; BCD1,la cifra di ordine alto, viene tradotta in ASCII1 alla locazione 0300, mentre BCD2, la cifra di ordine basso, viene tradotta in ASCII2 alla locazione 0301. Questa differenza è dovuta al fatto che normalmente si preferisce stampare prima la cifra di ordine alto.
Quindi, la stringa ASCII che inizia alla locazione 0300 può essere inviata pari pari ad una stampante e apparire così nel normale ordine "di stampa", con i digit di ordine alto che precedono i digit di ordine basso.

CAPITOLO 11

ISTRUZIONI ARITMETICHE E DI RICERCA DEI BLOCCHI

INTRODUZIONE

In questo capitolo tratteremo ancora il gruppo di Istruzioni Aritmetiche e Logiche a otto bit e il gruppo di istruzioni AF non specializzate. Ci interesseremo anche delle istruzioni aritmetiche a 16 bit e del gruppo delle utili istruzioni di ricerca dei blocchi. Questi quattro gruppi di istruzioni compaiono rispettivamente nelle tabelle 11-1, 11-2, 11-3, 11-4.
Alla fine di questo capitolo vi sarete fatti un'esperienza nell'uso di tutte le istruzioni che il microprocessor Z-80 può eseguire, ; vi mancano solo le istruzioni di input, output e interrupt, di cui parleremo dettagliatamente nel prossimo volume.

OBIETTIVI

Alla fine di questo capitolo sarete in grado di:

- Scrivere programmi per sommare, sottrarre, moltiplicare e dividere numeri interi binari a otto bit.
- Scrivere programmi per sommare, sottrarre, moltiplicare e dividere numeri interi binari a 16 bit.
- Capire e usare l'istruzione di aggiustamento decimale dell'accumulatore (DAA) e l'aritmetica BCD.
- Descrivere le funzioni dei flag H e N relativamente all'aritmetica BCD ed all'istruzione DAA.
- Capire ed usare le istruzioni di somma con riporto e sottrazione con riporto.
- Capire e usare l'istruzione di confronto, CP, e le istruzioni di ricerca dei blocchi, che ne costituiscono un ampliamento.

SORGENTE

	INDIRIZZAMENTO TRA REGISTRI							INDIR. TRAM. REG.	INDICIZZ		IMM
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
SOMMA 'ADD'	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	**C6 n**
SOMMA CON RIPORTO 'ADC'	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	**CE n**
SOTTRAZIONE 'SUB'	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	**D6 n**
SOTTRAZIONE CON RIPORTO 'SBC'	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	**DE n**
'AND'	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	**E6 n**
'XOR'	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	**EE n**
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	**F6 n**
CONFRONTO 'CP'	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	**FE n**
INCREMENTO 'INC'	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DECREMENTO 'DEC'	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

Tabella 11-1. Gruppo delle istruzioni aritmetiche e logiche a 8 bit.

Aggiustamento Decimale dell'Accumulatore 'DAA'	27
Complemento dell'Accumulatore "CPL"	2F
Negazione dell'Accumulatore 'NEG' (complemento a 2)	ED 44
Complemento del Flag di Riporto 'CCF'	3F
Set del Flag di Riporto 'SCF'	37

Tabella 11-2. Operazioni diverse sui registri A ed F.

			SORGENTE					
			BC	DE	HL	SP	IX	IY
DESTINAZIONE	SOMMA 'ADD'	HL	**09**	**19**	**29**	**39**		
		IX	DD 09	DD 19		DD 39	DD 29	
		IY	FD 09	FD 19		FD 39		FD 29
	SOMMA CON RIPORTO E SET DEI FLAGS 'ADC'	HL	ED 4A	ED 5A	ED 6A	ED 7A		
	SOTTRAZIONE CON RIPORTO E SET DEI FLAGS 'SBC'	HL	ED 42	ED 52	ED 62	ED 72		
	INCREMENTO 'INC'		**03**	**13**	**23**	**33**	DD 23	FD 23
	DECREMENTO 'DEC'		**0B**	**1B**	**2B**	**3B**	DD 2B	FD 2B

Tabella 11-3. Gruppo delle istruzioni aritmetiche a 16 bit.

LOCAZIONE DI RICERCA	
INDIR. TRAM. REG.	
(HL)	
ED A1	'CPI' Incrementa HL, Decrementa BC
ED B1	'CPIR', Incrementa HL, Decrementa BC Ripete finché BC = 0 o riscontra un confronto positivo
ED A9	'CPD' Decrementa HL & BC
ED B9	'CPDR' Decrementa HL & BC Ripete finché BC = 0 o riscontra un confronto positivo

Il Registro HL punta alla locazione di memoria che deve essere confrontata col contenuto dell'accumulatore.
Il Registro BC è il contatore di byte.

Tabella 11-4. Gruppo di ricerca di blocco.

GRUPPO ARITMETICO A OTTO BIT

Le istruzioni del gruppo aritmetico a otto bit riguardano tutte le operazioni di somma o di sottrazione di byte a otto bit. Le istruzioni INC e DEC sommano o sottraggono l'esadecimale 01 ad un registro o ad un byte di memoria. Le istruzioni ADD e SUB fanno sì che un byte di un registro o di una locazione di memoria sia sommato o sottratto al byte contenuto nell'accumulatore, il risultato essendo posto nell'accumulatore. Le istruzioni INC, DEC, ADD e SUB per le operazioni a otto bit alterano i flag nel seguente modo:

Flag di Zero: Se il risultato nell'accumulatore è zero, il flag Z viene settato, altrimenti è resettato.

Flag di Segno: Se il risultato nell'accumulatore è negativo (cioè il bit più significativo è a uno), il flag S è settato, altrimenti è resettato.

Flag di Carry: Se dalle operazioni INC, DEC, ADD o SUB risulta un riporto o un riporto negativo per/da un "fantomatico" nono bit, il flag C viene settato, altrimenti è resettato.

Flag di parità/ Overflow (P/V): Il flag P/V si comporta esattamente come un indicatore di overflow per l'aritmetica complemento a due. L'argomento è trattato nel Capitolo 8.

Flag H: Il flag di Half Carry (Carry di mezzo) viene settato se c'è un riporto o un riporto negativo, come risultato dell'addizione o della sottrazione del digit di ordine basso di due numeri BCD impaccati, altrimenti viene resettato. Parleremo di questo flag nel paragrafo riguardante l'istruzione DAA.

Flag N: Il Flag di sottrazione (N) viene settato per tutte le operazioni di sottrazione e resettato per tutte le operazioni di addizione. Anche di questo flag parleremo nel paragrafo riguardante l'istruzione DAA.

Eccovi parecchi esempi che illustrano le istruzioni INC, DEC, ADD e SUB. Il simbolo X nelle colonne relative di flag indica che il flag stesso non è modificato dall'operazione.

Istruzione	*Accumulatore Prima della esecuzione*	*Accumulatore Dopo la esecuzione*	*Flag dopo l'esecuzione* S	Z	H	P/V	N	C
INC A	04	05	0	0	0	0	0	X
INC A	FF	00	0	1	1	0	0	X
DEC A	00	FF	1	0	1	0	1	X
ADD A,80H	00	80	0	0	0	0	0	0
ADD A,80H	80	F0	1	0	0	1	0	0
ADD A,F0H	F0	E0	1	0	0	0	0	1
ADD A,11H	22	33	0	0	0	0	0	0
ADD A,18H	29	41	0	0	1	0	0	0
ADD A,94H	93	27	0	0	0	1	0	1
ADD A,99H	99	32	0	0	1	1	0	1
SUB 33H	33	00	0	1	0	0	1	0
SUB 02H	10	0E	0	0	1	0	1	0
SUB 22H	10	EE	1	0	1	0	1	1

Per capire gli esempi sopra riportati, eseguite le operazioni aritmetiche binarie indicate e applicate le regole già enunciate per settare e resettare i flag opportuni. Per esempio, prendiamo in considerazione l'istruzione

ADD A,18H

dove l'accumulatore contiene già 29H. Si può quindi scrivere l'addizione binaria:

```
  0 0 1 0 1 0 0 1     Accumulatore
+ 0 0 0 1 1 0 0 0
-----------------
  0 1 0 0 0 0 0 1
```

Flag S: 0 perchè il bit D7 è zero
Flag Z: 0 perchè il risultato non è zero
Flag H: 1 perchè c'è stato un riporto dal bit di D3 al bit D4 durante l'addizione bit per bit
Flag P/V: 0 perchè la somma dei due numeri complemento a due positivi, dà un numero complemento a due pure positivo
Flag N: 0 perchè l'operazione è un'addizione
Flag C: 0 perchè non c'è stato riporto dal bit D7

Le istruzioni ADC a otto bit, "somma con riporto", e SBC, "sottrazione con riporto", eseguono un'operazione costituita da tre momenti:

Passo 1: Somma o sottrae il byte indicato con l'accumulatore, come nel caso di un'istruzione ADD o SUB. Non si modifica nessuno dei bit dei flag.

Passo 2: Somma o sottrae il flag C con l'accumulatore. Ovvero, se il flag C era a 1 prima dell'esecuzione dell'istruzione ADC o SBC, somma o sottrae 01 dal byte nell'accumulatore.

Passo 3: Esegue l'aggiustamento dei flag, basandosi sui risultati dei due passi precedenti.

Vi diamo alcuni esempi che illustrano le istruzioni ADC e SBC.

Istruzione	*Accumulatore prima dell'esec.*	*Flag C*	*Accumulatore dopo l'esecuzione*	*Flag dopo l'esecuzione* S	Z	H	P/V	N	C
ADC A,00H	01	1	02	0	0	0	0	0	0
ADC A,00H	01	0	01	0	0	0	0	0	0
ADC A,90H	97	1	28	0	0	0	1	0	1
ADC A,19H	39	1	53	0	0	1	0	0	0
SBC A,00H	00	1	FF	1	0	1	0	1	1
SBC A,01H	00	1	00	0	1	0	0	1	0
SBC A,80H	00	1	7F	1	0	1	1	1	1

Le istruzioni ADC e SBC sono particolarmente utili nelle operazioni aritmetiche binarie a più byte. Consideriamo il programma indicato qui di seguito, che esegue una somma a *multipla precisione* a più byte tra due numeri binari immagazzinati in memoria. Il numero massimo di byte per ogni addendo è memorizzato nel registro C. Le stringhe di byte, che costituiscono il primo e il secondo addendo, iniziano con i byte meno significativi posti alle locazioni di memoria puntate rispettivamente dai registri HL e IX. IY contiene l'indirizzo di partenza (byte meno significativo) della stringa risultato.

```
        LD C,08H        ;Otto byte in ogni addendo
        LD HL,0200H     ;Addendo 1
        LD IX,0210H     ;Addendo 2
        LD IY,0220H     ;Somma
        SUB A           ;Azzera l'accumulatore e il flag C
ADDB:   LD A,(HL)       ;Prendi un byte della stringa 1
        ADC A,(IX)      ;Somma un byte della stringa 2
        LD (IY),A       ;Memorizza la somma nella stringa 3
        INC HL          ;Aggiorna i puntatori di memoria
        INC IX
        INC IY
        DEC C           ;Elaborati tutti i byte?
        JR NZ,ADDB      ;Se no, somma i byte successivi
        JR C,ERROR      ;Se sì, controlla il carry. Overflow
                        ;se C = 1
        RST 38H         ;Restituisci il controllo al sistema operativo
```

Vediamo come opera tale programma. La Figura 11-1 mostra come sono disposti gli addendi e il risultato nelle loro rispettive locazioni di memoria, e come vengono manipolati dal programma. Sono anche riportati i valori iniziali di HL,IX e IY. Questi registri vengono aggiornati man mano che vengono sommate le successive coppie di byte. Notate come, in caso di riporto, questo viene sommato da parte dell'istruzione ADC al byte di ordine alto che segue. L'istruzione SUB A inizialmente azzera il flag C, in modo che la prima somma è equivalente ad un'istruzione ADD. Dopo che tutte le coppie di byte della stringa 1 e della stringa 2 sono state sommate, viene fatto un controllo per individuare la presenza di un riporto, da parte dell'istruzione JR C,ERROR. L'istruzione ADC A,(IX) è stata l'ultima che ha potuto alterare il flag C, cosicché esso sta ancora ad indicare se si è verificato un overflow nella somma degli ultimi due byte (quelli più significativi). ERROR rappresenta la locazione di memoria di una routine che gestisce ed eventualmente stampa messaggi di overflow, e che qui non è riportata.

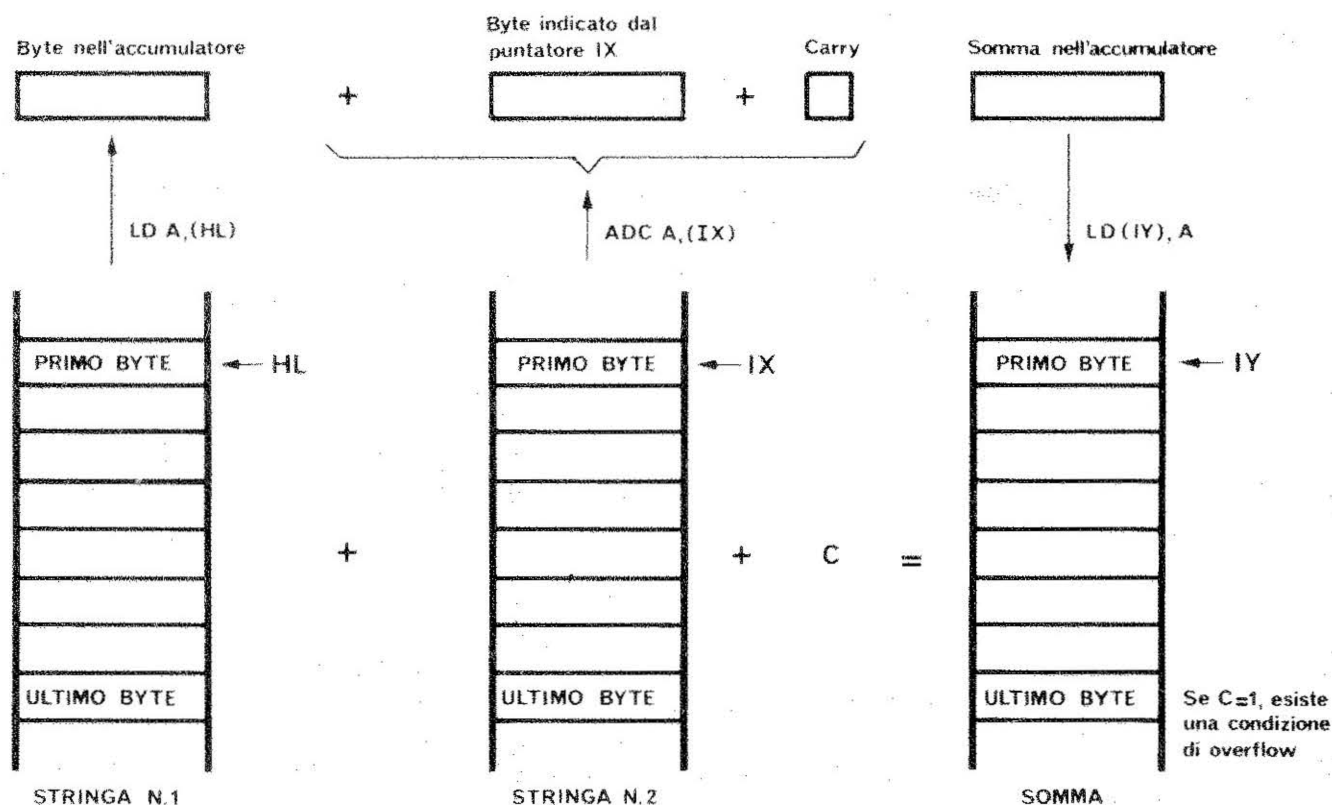


Figura 11-1

Il programma che vi abbiamo mostrato si può applicare anche alla somma di più byte in complemento a due, ammesso che i due addendi siano numeri in complemento a due di n-bit, in cui n=8 (lunghezza in byte dei due addendi). Nella somma in complemento a due, è pero necessario controllare come indicatore di overflow, il flag P/V, e non più il flag C come nel caso precedente.

L'ISTRUZIONE DAA

Per l'aritmetica decimale codificata binaria (BCD), è necessaria un'istruzione particolare che converta i risultati binari in risultati BCD. Lo Z-80 è in grado di eseguire direttamente solo addizioni e sottrazioni binarie. Dato che l'addizione e la sottrazione binaria sono essenzialmente identiche a quelle in complemento a due(a parte il modo di segnalare l'overflow), lo Z-80 puo eseguire anche l'aritmetica complemento a due. L'aritmetica decimale, invece non è uguale all'aritmetica binaria. Consideriamo il seguente problema di addizione, risolto prima usando dei numeri binari e poi dei numeri byte BCD impaccati:

```
 0 0 0 0 1 0 0 0 =  8 (base 10) = 08 (BCD impaccati)
+0 0 0 0 1 0 0 1 =  9 (base 10) = 09 (BCD impaccati)
-----------------
 0 0 0 1 0 0 0 1 = 17 (base 10) = 11 (BCD impaccati)
```

Notate che il risultato, interpretato come numero binario, è corretto, ma non lo è se lo interpretiamo come numero BCD impaccato. La ragione di questo sta nel fatto che le basi dei numeri sono diverse. Lo Z-80 considera i due digit di un numero BCD impaccato come due digit *esadecimali* perchè quattro bit, nella rappresentazione binaria, possono rappresentare 16 valori diversi.
Quindi, durante un'operazione aritmetica come l'addizione, si ha un riporto dai quattro bit (nibble) di destra al nibble di sinistra quando la somma è invece maggiore di 16. Per l'addizione BCD, si dovrebbe avere questo riporto quando la somma è maggiore di 10. Perciò l'addizione BCD e quella binaria non producono lo stesso risultato quando:

a. La somma di due nibble a quattro bit sta fra 10 e 15 (compresi), ad es.:

```
  1 0 0 1   9
 +0 0 1 0   2
 ------------
  1 0 1 1   B     come somma esadecimale;
                  sarebbe = 11 come somma BCD impaccata
```

In questo caso, l'addizione binaria non genera alcun riporto verso il nibble seguente, mentre l'addizione decimale lo produrrebbe.

b. La somma di due digit a quattro bit è maggiore o uguale a 16, ad es.:

```
  1 0 0 1    9
 +1 0 0 1    9
 -------------
 1 0 0 1 0  12    come somma esadecimale;
                  sarebbe = 18 come somma BCD impaccata
```

In questo caso l'addizione binaria genera un riporto verso il nibble seguente; esiste però ancora una differenza di 6 tra i due risultati.

In entrambi i casi presi in considerazione, il risultato esadecimale meno il risultato decimale (letto come se fosse un dato esadecimale) dà 6. Se ne può dedurre che quando si verifica (a) o (b) questo è sempre vero. Perciò la CPU dello Z-80 ha un'istruzione speciale, l'istruzione di *aggiustamento decimale dell'accumulatore*, DAA, che può rivelare se si verifica (a) o (b) e, in tal caso, sommare 6 ad un nibble.
Il modo per rivelarlo è molto semplice. Quando si verifica un caso del tipo (a), ci si ritrova

con un nibble di risultato che contiene un valore superiore a 9 (in esadecimale: A, B, C, D, E o F): è quindi necessario sommare 6 a tale valore.
Quando si verifica un caso del tipo (b), invece la somma dei due nibble genera un riporto. Tale riporto sarà segnalato nel flag H o nel flag C, a seconda che il riporto sia stato rispettivamente generato dalla somma dei due nibble più leggeri o dalla somma dei due nibble più pesanti (dei due byte addendi).
Consideriamo la seguente sequenza di istruzioni che esegue un'addizione BCD impaccata tra il contenuto dell'accumulatore e il registro B:

```
ADD A, B
DAA
```

Vediamo come operano queste istruzioni su cinque insiemi diversi di dati:

	1	2	3	4	5
Registro B	11	19	91	99	09
Accumulatore prima di ADD A, B	22	18	81	88	05
Dopo ADD A, B: Accumulatore	33	31	12	21	0E
Flag H	0	1	0	1	0
Flag C	0	0	1	1	0
Accumulatore dopo DAA	33	37	72	87	14

La terza e quarta coppia di valori (91 e 81; 99 e 88) danno una somma il cui risultato è maggiore di 99 che è il massimo valore BCD impaccato che l'accumulatore può contenere. Poiché manca un terzo nibble per rappresentare le centinaia, l'1 di riporto, delle decine, non può essere posto nell'accumulatore (overflow), ed è indicato solo dal fatto che il flag C è posto a 1. Notate che le risposte esatte sono 172 e 187. L'ultima coppia di dati (09 e 05) è un esempio del verificarsi della condizione (a) già descritta. Non vengono settati nè H nè C, ma è richiesto comunque un aggiustamento dato che E non è un numero decimale.
Quindi, per convertire il programma che somma numeri binari a più byte in un programma che somma numeri BCD impaccati a più byte, è necessaria solo una modifica. Inserite semplicemente un'istruzione DAA fra le istruzioni ADC A, (IX) e LD (IY),A.

ISTRUZIONI ARITMETICHE A SEDICI BIT

Alle istruzioni ADD,ADC,SBC,INC e DEC, che operano su di un byte, corrispondono delle istruzioni analoghe che eseguono essenzialmente la stessa operazione usando però coppie di registri a 16 bit. Tali istruzioni, nonchè i codici esadecimali ad esse relativi, sono riportate nella Tabella 11-3. Nella Tabella 11-5 potete vedere in dettaglio come i flag vengono manipolati dalle istruzioni aritmetiche a 16 bit. Come potete vedere, essa contiene numerose informazioni che non compaiono nella Tabella 11-3. Nel pieghevole allegato, descritto nell'Appendice 1, vi presentiamo un gruppo di tabelle simili che coprono tutto il set di istruzioni dello Z-80.

LE ISTRUZIONI DI CONFRONTO E DI RICERCA DEI BLOCCHI: CPI, CPD, CPIR e CPDR

L'istruzione CP confronta il contenuto dell'accumulatore con un certo byte operando indicato con s, elabora la differenza A-s, e altera i flag C,Z,P/V (come flag di overflow), S e H coerentemente con il risultato della sottrazione. Il flag N è settato in quanto si ha una operazione di sottrazione. Notate che nè l'accumulatore nè il byte s vengono modificati dall'operazione CP. La differenza, A-s, è presente internamente alla CPU, ma l'unico effetto, visibile al programma, dell'istruzione CP è quello di alterare i flag. Quindi, per esempio, l'istruzione CP B ha sui registri A, B, e F un effetto del tutto analogo a quello che ha la sequenza:

```
LD C,A      Salva l'accumulatore
SUB B       Esegui la sottrazione modificando i flag
LD A,C      Ripristina l'accumulatore
```

Codice Mnemonico	Operazione Simbolica	Flags S	Z		H		P/V	N	C	Codice Operativo 76 543 210	Hex	N° di Bytes	N° di Cicli M	N° di Stati T	Commenti	
ADD HL, ss	HL ← HL+ss	•	•	X	X	X	•	0	↕	00 ss1 001		1	3	11	ss	Reg.
															00	BC
ADC HL, ss	HL ← HL+ss+CY	↕	↕	X	X	X	V	0	↕	11 101 101	ED	2	4	15	01	DE
										01 ss1 010					10	HL
															11	SP
SBC HL, ss	HL ← HL-ss-CY	↕	↕	X	X	X	V	1	↕	11 101 101	ED	2	4	15		
										01 ss0 010						
ADD IX, pp	IX ← IX + pp	•	•	X	X	X	•	0	↕	11 011 101	DD	2	4	15	pp	Reg.
										00 pp1 001					00	BC
															01	DE
															10	IX
															11	SP
ADD IY, rr	IY ← IY + rr	•	•	X	X	X	•	0	↕	11 111 101	FD	2	4	15	rr	Reg.
										00 rr1 001					00	BC
															01	DE
															10	IY
															11	SP
INC ss	ss ← ss + 1	•	•	X	•	X	•	•	•	00 ss0 011		1	1	6		
INC IX	IX ← IX + 1	•	•	X	•	X	•	•	•	11 011 101	DD	2	2	10		
										00 100 011	23					
INC IY	IY ← IY + 1	•	•	X	•	X	•	•	•	11 111 101	FD	2	2	10		
										00 100 011	23					
DEC ss	ss ← ss - 1	•	•	X	•	X	•	•	•	00 ss1 011		1	1	6		
DEC IX	IX ← IX - 1	•	•	X	•	X	•	•	•	11 011 101	DD	2	2	10		
										00 101 011	2B					
DEC IY	IY ← IY - 1	•	•	X	•	X	•	•	•	11 111 101	FD	2	2	10		
										00 101 011	2B					

Tabella 11-5. Gruppo delle istruzioni aritmetiche a 16 bit.

Avete già visto un'applicazione dell'istruzione CP nell'esperimento che illustra le tabelle di salto alla fine del Capitolo 7.
Le istruzioni di ricerca dei blocchi che compaiono nella Tabella 11-4 operano in modo analogo alle istruzioni di trasferimento di blocchi di dati. Come le istruzioni di trasferimento blocchi "derivano" dall'istruzione base LD, così le istruzioni di ricerca blocchi sono l'estensione dell'istruzione base CP. Queste ultime istruzioni facilitano la ricerca in locazioni di memoria contigue, di un dato che coincide con un "byte chiave" contenuto nell'accumulatore. Come per le istruzioni di trasferimento di blocchi, alcuni registri devono essere inizializzati prima dell'esecuzione di qualunque istruzione di ricerca dei blocchi:

BC = numero delle locazioni di memoria in cui effettuare la ricerca
HL = indirizzo del primo byte da confrontare con il contenuto dell'accumulatore
A = il valore chiave che deve essere confrontato con i byte di memoria del blocco

L'esecuzione dell'istruzione CPI, "confronta-incrementa" fa sì che si verifichi quanto segue:

1. Il byte nella locazione indirizzata dalla coppia di registri HL è confrontato con il contenuto dell'accumulatore. I flag Z,S e H sono alterati in base al risultato del confronto. L'istruzione CP (HL) coinvolge anche il flag C, mentre l'istruzione CPI (come le altre istruzioni di ricerca dei blocchi) non lo altera.
2. Il contenuto della coppia di registri HL viene incrementato.
3. Il contenuto della coppia di registri BC viene decrementato. A questo punto, viene settato il flag Z se A = (HL), altrimenti tale flag è resettato. Il flag P/V è resettato se la coppia di registri BC = 0000, altrimenti è settato.

L'esecuzione istruzione CPIR, "confronta-incrementa-ripeti", comporta invece che:

1. Il byte contenuto nella locazione indirizzata dalla coppia di registri HL è confrontato con il contenuto dell'accumulatore. I flag Z,S e H sono alterati coerentemente con il risultato.
2. Il contenuto della coppia di registri HL viene incrementato.
3. Il contenuto della coppia di registri BC viene decrementato. Il flag Z è stato settato se A=(HL), altrimenti è settato.
4. Se la coppia di registri BC=0000 o se A=(HL), l'esecuzione dell'istruzione termina e il programma continua con l'istruzione successiva. Altrimenti, l'istruzione viene ripetuta (passi 1, 2, 3 e 4).

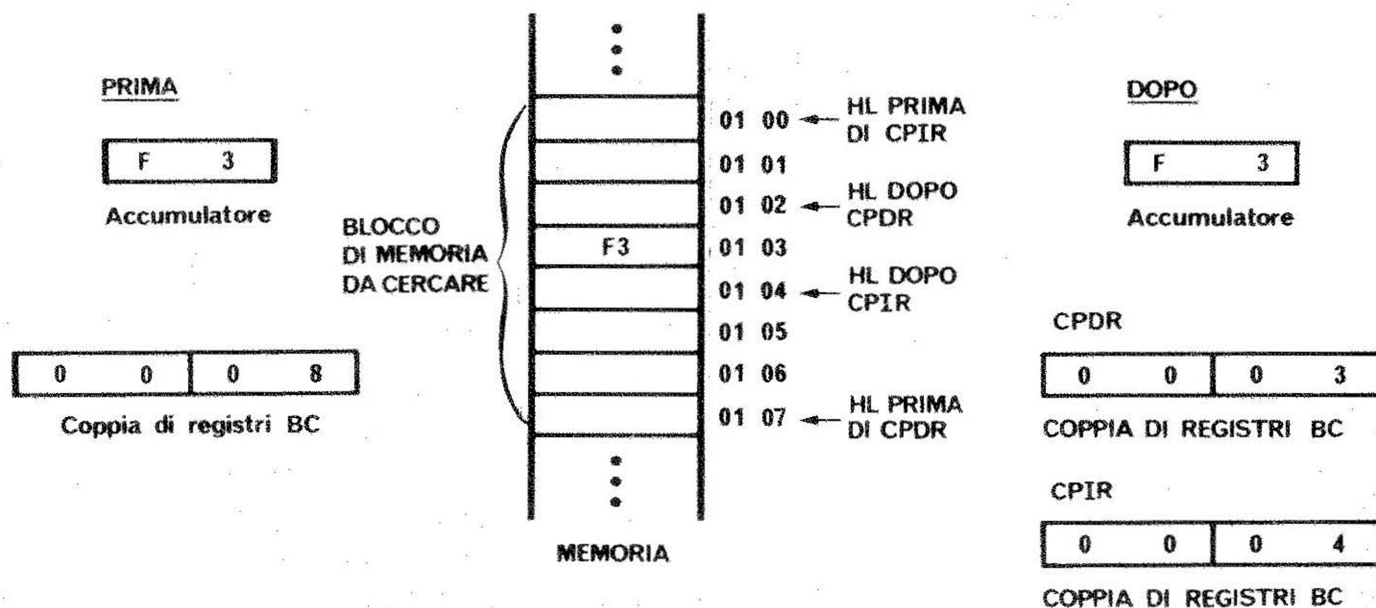


Figura 11-2

Le istruzioni CPD, "confronta-decrementa", e CPDR, "confronta-decrementa-ripeti", sono molto simili alle istruzioni precedenti: l'unica differenza sta nel fatto che il Passo 2 decrementa HL anziché incrementarlo. La Figura 11-2 illustra i registri e le locazioni di memoria prima e dopo l'esecuzione delle istruzioni CPIR e CPDR.

INTRODUZIONE AGLI ESPERIMENTI

I seguenti esperimenti vi presentano dei programmi-tipo che effettuano le operazioni aritmetiche fondamentali.
Vengono prese in esame anche alcune applicazioni delle istruzioni di ricerca blocchi. Gli esperimenti che eseguirete si possono così riassumere:

Esperimento N.	Commenti
1	Illustra un programma di moltiplicazione binaria a multipla precisione
2	Illustra un programma di sottrazione BCD a più byte.
3	Illustra un programma di divisione in cui un numero binario a 16 bit viene diviso per un numero binario a 8 bit.
4	Illustra due utili applicazioni delle istruzioni di ricerca blocchi e di confronto.

ESPERIMENTO N. 1

Scopo

Questo esperimento presenta un metodo per scrivere un programma che operi con l'aritmetica binaria in multipla precisione. Il programma che vi presentiamo qui di seguito moltiplica due numeri binari a 16 bit memorizzati alle locazioni 0130-0131 e 0132-0133 e salvando il risultato nelle locazioni 0134-0135. E' poi presentato un programma che moltiplica due numeri binari di n byte (con n qualunque, ma uguale per entrambi gli operandi) per ottenere un prodotto della stessa lunghezza.
Vedrete quindi che il vostro Nanocomputer può moltiplicare numeri a 64 bit, esattamente come i computer più grossi. (Ovviamente la velocità *non* è la stessa).

Programma N. 32

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
0100	21 00 00	MLT16:	LD HL,0000H	;HL=prodotto, inizializzato a 0
0103	ED 5B 30 01		LD DE,(0130H)	;DE=moltiplicando
0107	ED 4B 32 01		LD BC,(0132H)	;BC=moltiplicatore
010B	7A		LD A,D	;Controlla se DE=0
010C	B3		OR E	
010D	20 01		JR NZ,MLT	;DE=0 (Flag Z settato)?
010F	FF		RST 38H	;Sì, ritorna al sistema operativo del ;Nanocomputer
0110	CB 38	MLT:	SRL B	;BC: sposta verso destra, con ;introduzione di zero
0112	CB 19		RR C	;Il flag C è uguale al bit da moltiplicare
0114	30 04		JR NC,NCF	;Controlla il flag C
0116	19		ADD HL,DE	;Il flag C è settato, somma DE a HL
0117	30 01		JR NC,NCF	;ADD ha generato un riporto (Flag ;C settato)?

0119	FF		RST 38H	;Sì, overflow, ritorna al sistema ;operativo del Nanocomputer
011A	78	NCF:	LD A,B	;Se il flag C è resettato, controlla se ;BC=0
011B	B1		OR C	
011C	CA 29 01		JP Z,ANS	;Se BC=0, memorizza la risposta
011F	CB 23		SLA E	;Altrimenti ruotare DE verso sinistra
0121	CB 12		RL D	
0123	30 01		JR NC,CONT	;Esiste overflow (Flag C settato)?
0125	FF		RST 38H	;Sì, ritorna al sistema operativo del ;Nanocomputer
0126	C3 10 01	CONT:	JP MLT	;No, quindi continua
0129	22 34 01	ANS:	LD (0134H),HL	;Memorizza la risposta
012C	FF		RST 38H	;Restituisci il controllo al sistema ;operativo

Passo 1

Parliamo brevemente della metodologia usata nel programma, per eseguire la moltiplicazione binaria. Per semplicità, limitiamoci a moltiplicare due numeri binari a quattro bit. I principi che vi esponiamo per la moltiplicazione a quattro bit, si applicano anche a quella ad otto bit, sedici bit, o qualunque altra moltiplicazione binaria.
Supponiamo di voler moltiplicare 0011 per 0101. Il procedimento, molto simile a quello che userete normalmente nelle moltiplicazioni decimali, è il seguente:

```
      0 0 1 1     Moltiplicando
    × 0 1 0 1     Moltiplicatore
    ---------
      0 0 1 1     1 X 0 0 1 1
    0 0 0 0 -     0 X 0 0 1 1
  0 0 1 1 - -     1 X 0 0 1 1
0 0 0 0 - - -     0 X 0 0 1 1
-------------
0 0 0 1 1 1 1
```

Il procedimento quindi è molto semplice: ogni volta che nel moltiplicatore è presente un bit a 1, questo fatto provoca lo shift del moltiplicando verso sinistra in modo da dare un prodotto parziale. La somma di tutti i prodotti parziali dà il risultato della moltiplicazione. Gli shift, naturalmente, introducono degli zeri (non indicati) sulla destra.
Tale programma perciò si serve della tecnica di shift e di addizione (SHIFT AND ADD).
La coppia di registri HL contiene il prodotto, mentre BC contiene il moltiplicatore e DE il moltiplicando.

Passo 2

Caricate ed eseguite il programma con diversi di numeri binari a 16 bit. Notate che, finché i numeri restano relativamente piccoli, il risultato è giusto. Per esempio,

0400 x 0020 = 8000
00FF x 00FF = FE01
0100 x 00FF = FF00

Che cosa succede se abbiamo 0100 x 0100?

Otteniamo 0000 nel registro HL, che è chiaramente un risultato sbagliato. Tentiamo di fare il calcolo a mano. La risposta è 1 seguito da 16 zero, o, in tre byte esadecimali,

01 00 00.

Ma, purtroppo, abbiamo solo due byte disponibili per il risultato, quindi il byte tre, il byte più significativo, non può comparire nella risposta stessa. Si tratta di un caso di OVERFLOW che viene rivelato nel programma dalle istruzioni JR C, NCF e JR C,CONT, che, nel caso specifico provocano l'esecuzione di RST 38H. Nel nostro programma abbiamo, in caso di overflow, generato quindi un salto al sistema operativo; in forma più generale, verrà effettuato un salto ad una routine di gestione dell'errore. Notate che l'overflow viene controllato in due punti del programma.
Vediamo due esempi con numeri di quattro bit per vedere perchè questo si verifica.

```
            1 0 0 0  Moltiplicando
          × 0 1 1 0  Moltiplicatore
          ---------
            0 0 0 0
          1 0 0 0    Si verifica overflow perchè il flag di Carry è settato
                     già dallo shift del moltiplicando.
                     Un flag di Carry settato indica che il prodotto contiene
                     più di quattro bit.

            0 1 1 0  Moltiplicando
            0 0 1 1  Moltiplicatore
          ---------
            0 1 1 0
          0 1 1 0
Somma parziale  1 0 0 1 0  Si verifica overflow perchè l'operazione Shift e Somma fa sì
                     che il flag di Carry venga settato, indicando così che il
                     prodotto contiene più di quattro bit.
```

Quindi, per rivelare l'overflow nelle due situazioni prese in esame, è sufficiente che il flag di Carry venga controllato dopo lo shift del moltiplicando (coppia di registri DE) e dopo che si è effettuata una somma parziale (registro HL) o si è giunti al risultato.

Passo 3

Le stesse tecniche usate per la moltiplicazione a 16 bit (2 byte) si possono applicare alla moltiplicazione di numeri binari a n-byte, dove n è qualunque numero intero positivo. La sequenza di istruzioni che vi mostriamo ora, moltiplica due numeri binari formati da un numero NUM di byte, che sono memorizzati in un numero NUM di locazioni di memoria successive, dove il byte meno significativo inizia agli indirizzi YNUM e XNUM, rispettivamente per il moltiplicando ed il moltiplicatore, il prodotto è memorizzato in NUM locazioni di memoria successive con il byte meno significativo all'indirizzo ZNUM.

```
MLTN:    LD B,NUM        ;Carica il numero di byte per numero nel registro B
                         ;(NUM= costante esadecimale ad un byte)
         LD HL,ZNUM      ;Memorizza in HL l'indirizzo del risultato del
                         ;prodotto (ZNUM= costante esadecimale a
                         ;a due byte)
INIT:    LD (HL),00H     ;Memorizza 00 in tutta la stringa del risultato
         INC HL
         DJNZ INIT
SHIFTX:  LD B,NUM        ;Sposta il moltiplicatore XNUM di un bit verso destra
                         ;e trasferisci nel registro B il numero dei byte
                         ;degli operandi
```

```
         LD IX,XNUM+NUM-01H    ;IX= indirizzo del byte più significativo del
                               ;moltiplicatore (XNUM= costante esadecimale a
                               ;due byte)
         XOR A
RTX:     RR (IX)
         DEC IX
         DJNZ RTX
         JR NC,ZERO            ;Se non c'è riporto, il bit attualmente nel moltiplicatore
                               ;è zero, quindi non effettuare la somma tra il
                               ;moltiplicando e il prodotto parziale
         LD B,NUM              ;Se il bit di Carry è settato, il bit attualmente nel
                               ;moltiplicatore è 1, quindi si somma il
                               ;moltiplicando al prodotto parziale che risiede nella
                               ;locazione ZNUM
         LD HL,ZNUM
         LD IY,YNUM            ;YNUM= costante esadecimale a due byte
         CCF                   ;Complementa il flag di Carry che è stato settato da
                               ;queste istruzioni. Il flag C è ora zero.
NXTBYT:  LD A,(IY)             ;Somma il moltiplicando al prodotto parziale
         ADC A,(HL)
         LD (HL),A             ;Memorizza la somma nella stringa del risultato
         INC IY                ;Aggiorna il puntatore dei byte
         INC HL
         DJNZ NXTBYT
ZERO:    LD B,NUM              ;Controlla se il moltiplicatore è zero.
                               ;Se lo è, l'operazione è terminata
         LD IX,XNUM
         XOR A                 ;Azzera A per il test di zero
ZCHK:    OR (IX)               ;Controlla se il moltiplicatore = 0
         JR NZ,SHIFTY          ;Se il moltiplicatore non è zero, effettua ancora lo shift
                               ;del moltiplicando
         INC IX
         DJNZ ZCHK
         RST 38H               ;Restituisci il controllo al sistema operativo,
                               ;se il moltiplicatore è zero
SHITFY:  LD B,NUM              ;Shift del moltiplicando YNUM di un bit verso destra,
                               ;metti zero nel bit che si è liberato
         LD IY,YNUM
         XOR A                 ;Azzera il bit di Carry
LEFTY:   RL (IY)               ;Inizia lo shift
         INC IY
         DJNZ LFTY
         JP SHIFTX
         RST 38H               ;Il moltiplicatore = 0, quindi l'operazione è finita.
```

Rivedete con attenzione questa routine per essere sicuri di averla capita bene. Abbiamo usato per la prima volta variabili come NUM, XNUM, YNUM e ZNUM per indicare costanti ad uno e a due byte. Si tratta di un'abitudine molto diffusa nel software, quindi dovreste abituarvici.

Passo 4

Eseguite l'assemblaggio manuale del programma di moltiplicazione NUM-byte, sostituendo i vostri valori con NUM,XNUM,YNUM e ZNUM. Eseguite dei programmi di prova.

Passo 5

Notate che il programma di moltiplicazione NUM-byte non controlla la presenza o meno di overflow. Inserite voi stessi dei controlli e verificate che funzionino veramente.

ESPERIMENTO N. 2

Scopo

Questo esperimento presenta un programma che effettua sottrazioni BCD a più byte. Il programma che vi presentiamo ora prende due numeri BCD di un numero NUM di byte, i cui byte meno significativi si trovano alle locazioni XNUM e YNUM rispettivamente, quindi elabora la differenza fra questi numeri (la stringa che parte dall'indirizzo YNUM è sottratta alla stringa che inizia all'indirizzo XNUM), e memorizza il risultato della differenza in una stringa che parte dalla locazione ZNUM.

Programma N. 33

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
0200	06 N	SUBN:	LD B,NUM	;Il registro B conta il numero dei byte
0202	DD 21 X2 X1		LD IX,XNUM	
0206	FD 21 Y2 Y1		LD IY,YNUM	
020A	21 Z2 Z1		LD HL,ZNUM	
020D	37		SCF	;Setta il flag di Carry per il complemento ;a 100 per la prima sottrazione
020E	3E 99	NXTBY:	LD A,99H	;Trova cioé il complemento a 100 o a 99 ;del sottraendo
0210	CE 00		ADC A,00H	
0212	FD 96 00		SUB (IY)	
0215	DD 86 00		ADD A,(IX)	;Somma il byte del minuendo
0218	27		DAA	;Esegui l'aggiustamento per l'aritmetica ;decimale
0219	77		LD (HL),A	
021A	DD 23		INC IX	;Aggiorna i puntatori
021C	FD 23		INC IY	
021E	23		INC HL	
021F	10 ED		DJNZ NXTBYT	;Continua a sottrarre finchè vengono ;elaborati tutti i byte
0221	FF		RST 38H	;Ritorna al sistema operativo. L'assenza ;di riporto dopo l'ultimo byte indica ;overflow.

Passo 1

Vediamo come il programma funziona. Ricordate che, parlando della sottrazione di numeri complemento a due, abbiamo detto che la sottrazione di un numero in complemento a due equivale a formare il suo complemento a due e ad effettuare poi una somma utilizzando tale valore. La stessa cosa vale per la sottrazione BCD impaccata, solo che, anzichè formare un complemento a due, dovete formare un complemento a 100. Eccovi alcuni esempi:

Byte BCD impaccato	03	94	30	01	50
Complemento a 100	97	06	70	99	50

Quindi per trovare il complemento a 100 di un byte BCD impaccato, dovete semplicemente sottrarre ogni byte da 100 considerato come numero decimale a due digit. Analogamente, potete calcolare il complemento a 10 di un digit BCD. Eseguiamo ora una sottrazione BCD con quattro byte usando la tecnica "complementa e somma". Trovate la differenza: 256925 − 133639:

Passo 1: Formate il complemento a 100 del byte meno significativo del sottraendo:
100 − 39 = 61

Passo 2: Sommate 61 al byte meno significativo del minuendo (addizione decimale):
25 + 61 = 86
Non c'è riporto.

Passo 3: Dato che dal Passo 2 non è risultato alcun riporto, formate il complemento a 99 del successivo byte meno significativo del sottraendo:
99 − 36 = 63

Passo 4: Sommate 63 al successivo byte meno significativo del minuendo:
69 + 63 = 32
C'è riporto.

Passo 5: Dato che dal Passo 4 risulta un riporto, formate il complemento a 100 del byte più significativo del sottraendo: 100 − 13 = 87

Passo 6: Sommate 87 al byte più significativo del minuendo:
25 + 87 = 12 C'è riporto.

Passo 7: Il risultato è 113286, ed è corretto. Possiamo esserne sicuri perchè c'è stato un riporto. Se non ci fosse stato riporto, la risposta sarebbe stata sbagliata, e si sarebbe verificato l'overflow. Dato che i numeri BCD sono sempre maggiori o uguali a zero, l'overflow si verifica ogni volta che il sottraendo è maggiore del minuendo.

Non ci soffermeremo a dimostrare matematicamente la tecnica che abbiamo applicato. Per quanto riguarda il problema di sottrarre due numeri, in rappresentazione BCD impaccata, contenuti in più byte, è quindi sufficiente dire che per ogni byte, bisogna fare il complemento a due del sottraendo quando la sottrazione dei due byte precedenti (più leggeri) *non* ha generato un riporto negativo (non ha cioé richiesto un prestito).
E' necessario invece fare il complemento a 99 quando la sottrazione precedente ha generato un riporto negativo. Parlando della tecnica "complemento-somma":
Il riporto negativo nella sottrazione equivale a Carry = 0 nella procedura di complemento e addizione.

Analogamente, se il flag di Carry è 1 al termine della sottrazione BCD, non si verifica overflow. Questo fatto potrebbe, in un primo momento, sembrare "non intuitivo", ma, man mano che ci penserete, vi sembrerà sempre più naturale.

Passo 2

Caricate ed eseguite il programma provando con diversi valori BCD. Assicuratevi di definire i valori di:

NUM	=	costante esadecimale ad 1 byte che rappresenta il numero di byte BCD impaccati
XNUM	=	indirizzo a 2 byte del minuendo (esadecimale)
YNUM	=	indirizzo a 2 byte del sottraendo (esadecimale)
ZNUM	=	indirizzo a 2 byte del risultato della differenza (esadecimale)

ESPERIMENTO N. 3

Scopo

Questo esperimento presenta un programma che effettua un'operazione di divisione tra un numero binario a 16 bit e un numero binario a otto bit. Il risultato sarà costituito da quoziente e resto, entrambi su otto bit.
Nel programma, si suppone che, inizialmente, il registro HL contenga il dividendo binario a 16 bit, e che il registro D contenga il divisore binario a otto bit. Al termine dell'esecuzione, il quoziente a otto bit si trova nel registro L e il resto a otto bit si trova nel registro H. Perché l'algoritmo della divisione, che vedete qui implementato, operi correttamente, dobbiamo supporre che il divisore e il dividendo siano nella forma normalizzata. Ovvero:

a. Il bit più significativo del dividendo a 16 bit è zero, e

b. Il byte più significativo del dividendo è minore del divisore per essere sicuri che il quoziente potrà essere contenuto negli otto bit ad esso assegnati.

Programma N. 34

Locazione di memoria	Codice oggetto		Codice sorgente	Commenti
0300	06 08	DIV:	LD B,08H	;Numero di bit del divisore
0302	1E 00		LD E,00H	;Il divisore in DE
0304	29	NXTBIT:	ADD HL,HL	;Shift di HL verso sinistra, inserisci zero ;nel bit meno significativo
0305	AF		XOR A	;Resetta il flag di Carry
0306	ED 52		SBC HL,DE	;HL è maggiore di DE?
0308	23		INC HL	;Supponiamo di sì
0309	30 02		JR NC,NXT	;Altrimenti ripristina HL
030B	19		ADD HL,DE	;Somma di nuovo DE
030C	2B		DEC HL	;Poni a zero il bit del quoziente
030D	10 F5	NXT:	DJNZ NXTBIT	
030F	FF		RST 38H	

Passo 1

Vediamo prima e come opera il programma. L'algoritmo usato è molto simile al metodo usato quando si eseguono a mano delle lunghe divisioni decimali. Comunque, è più facile perchè sono coinvolti solo gli 0 e gli 1. Vediamo un esempio in cui il numero binario a quattro bit 8 divide il numero binario a otto bit 6E.

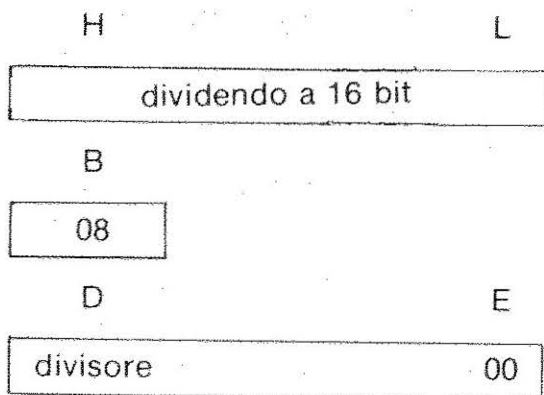
Impostate il problema come fareste con un normale problema di divisione, scrivendo tutti i numeri in codice binario:

0 1 1 0 1 1 1 0 : 1 0 0 0

Per determinare i bit successivi nel quoziente, inserite semplicemente nel quoziente un 1 se il divisore "sta" nella parte di dividendo su cui state lavorando, o uno 0 se esso non ci "sta". In realtà è più semplice vedere l'esempio:

```
0 1 1 0 1 1 1 0 : 1000 = 1101
— 1 0 0 0
---------
    1 0 1 1
  — 1 0 0 0
  ---------
    0 1 1 1 0
    — 1 0 0 0
    ---------
      0 1 1 0
```

Quindi il quoziente è 1101 = D (base 16) e il resto è 0110 = 6 (base 16).
Nel programma suddetto, i registro vengono inizializzati in questo modo:



Dato che il bit più significativo di HL è zero e il contenuto di D è maggiore del contenuto di H (vedi le ipotesi fatte a proposito della normalizzazione del dividendo e del divisore), il primo passo del programma è lo shift a sinistra, di HL (ADD HL, HL) seguito dal confronto del contenuto di DE e HL. Se DE è minore o uguale ad HL, nel quoziente viene posto un uno, altrimenti uno zero. Notate come il quoziente ruota nei quattro bit di ordine nasso della coppia di registri HL.

Questo è possibile perchè, man mano che viene inserito un nuovo bit nel quoziente, viene eliminato un bit del vecchio dividendo. Il metodo per determinare se il nuovo bit del quoziente (che è sempre il bit meno significativo del registro D) deve essere zero o uno, è sottrarre DE da HL. Un riporto indica che il bit del quoziente deve essere zero, mentre l'assenza di riporto implica che DE "sta" in HL, per cui il bit del quoziente è 1 (INC HL). Notate che la sottrazione HL-DE fa uso dell'istruzione SBC dato che non esiste un'istruzione SUB a 16 bit. C'è bisogno quindi dell'istruzione XOR A subito prima dell'istruzione SBC, per assicurarsi che il flag di Carry sia zero. Se DE non sta in HL (cioè: non è minore di HL), il contenuto di HL deve essere di nuovo ripristinato, addizionando ancora DE e decrementando HL.

Passo 2

Caricate ed eseguite il programma passo passo ed inserite diversi valori per gli operandi.

ESPERIMENTO N. 4

Scopo

Questo esperimento mostra due utili applicazioni delle istruzioni di ricerca blocchi e di confronto. Vi presentiamo due programmi che utilizzano queste istruzioni per eseguire delle funzioni molto usate.

Programma N. 35: Ricerca in una stringa di caratteri un carattere particolare.

Locazione di memoria	Codice oggetto	Codice sorgente	Commenti
0400	21 00 0A	LD HL,0A00H	;Indirizzo di inizio della stringa di caratteri
0403	01 20 00	LD BC,0020H	;Numero dei caratteri della stringa
0406	3E 24	LD A,24H	;Carattere da trovare: 24 è un "$" ;ASCII
0408	ED B1	CPIR	;Trova il $
040A	C2 0F 04	JP NZ,NOFIND	;Se il flag Z = 0, il carattere non è stato ;trovato
040D	2B	DEC HL	;Sottrai 1 da HL, in modo che punti ;alla posizione del carattere nella stringa
040E	03	INC BC	;Incrementa BC in modo che dia il numero ;del carattere nella stringa

Programma N. 36: Ricerca in una tabella di record un record particolare identificato da una stringa di 3 caratteri

0412	31 00 0F		LD SP,0FF0H	;Inizializza lo stack
0415	21 00 0C		LD HL,0C00H	;Indirizzo dell'ultimo record della tabella
0418	01 06 00		LD BC,0006H	;Numero di record nella tabella
041B	3A 00 0B		LD A,(0B00H)	;Primo carattere della stringa a 3 caratteri ;da identificare
041E	11 F9 FF		LD DE,FFF9H	;-(lungh. del record-1): complemento ;a due di 16 bit
0421	ED A9	NREC:	CPD	;Sono uguali?
0423	28 09		JR Z, CHECK	
0425	E2 42 04		JP PO,NOFIND	;Sono stati esaminati tutti i record?
0428	19	UPD:	ADD HL,DE	;Aggiorna HL all'inizio del record ;seguente
0429	3A 00 0B		LD A,(0B00H)	;Ricarica l'accumulatore
042C	18 F3		JR NREC	;Guarda il record successivo
042E	3A 01 0B	CHECK:	LD A,(0B01H)	;Confronta il secondo byte
0431	E5		PUSH HL	;HL punta ora all'ultimo byte del record ;che viene subito prima del record da ;controllare
0432	DD E1		POP IX	;Carica IX con HL
0434	DD BE 02		CP (IX+02H)	
0437	20 EF		JR NZ,UPD	;Ritorna indietro alla ricerca, se non ;trovi la corrispondenza
[illegible]	3A 02 CB		LD A,(0B02H)	;Confronta il terzo byte
[illegible]	DD BE 03		CP (IX+03H)	

```
043F    20 E7              JR NZ,UPD     ;Ritorna indietro alla ricerca se non trovi
                                         ;la corrispondenza
0441    23                 INC HL        ;HL punta ora al primo byte del record
                                         ;dove è stata trovata la corrispondenza,
                                         ;il flag Z è settato.
0442    FF        NOFIND:  RST 38H       ;Rientra nel sistema operativo.
```

Passo 1

Esaminiamo il Programma N. 35. Questo programma inizia memorizzando nella coppia di registri HL l'indirizzo del primo byte di una stringa di byte, nella coppia di registri BC il numero dei byte di questa stringa e nell'accumulatore il byte di chiave, cioè il byte di confronto. L'istruzione CPIR controlla sequenzialmente ogni byte della stringa finchè trova una corrispondenza o finchè non ci sono più byte da controllare. Se viene trovata una corrispondenza, l'istruzione CPIR setta il flag Z che fa sì che vengano eseguite le istruzioni DEC HL e INC BC, cosicchè HL punta proprio al byte della stringa di cui è stata verificata la uguaglianza e BC rappresenta il numero dello stesso nella stringa che lo contiene. Se non si trova una corrispondenza, il controllo è rimandato al sistema operativo con il flag Z e il flag P/V resettati, e BC=0000.
Questo programma, costruito come una subroutine, potrebbe essere usato per implementare le tabelle di salto, esattamente come quello che vi abbiamo mostrato nel Capitolo 7. I valori da ricercare possono essere memorizzati in una parte della memoria, separata da quella che contiene gli indirizzi di salto ad essi relativi. Il Programma N. 35 verrebbe chiamato quando si vogliono cercare dei valori ben precisi, dando come risultato di ritorno, nella coppia di registri BC, un indice di posizione del valore cercato. L'indice verrebbe quindi usato per trovare l'esatto indirizzo di salto per il successivo trasferimento del controllo.

Passo 2

Caricate ed eseguite il Programma N. 35. Usate diverse stringhe e diversi valori di chiave per controllare molto bene la logica di programma.

Passo 3

Esaminiamo ora il Programma N. 36. Supponiamo di avere posizionato in memoria la seguente tabella:

	NUMERO DEL BYTE								
Locazione	1	2	3	4	5	6	7	8	
0BD8	41	51	46	31	32	33	30	36	Record 1
0BE0	42	46	47	33	36	30	30	34	Record 2
0BE8	41	42	43	36	35	34	32	31	Record 3
0BF0	43	42	41	36	36	36	36	36	Record 4
0BF8	43	41	42	34	33	34	33	34	Record 5
0C00	42	42	42	31	32	33	32	31	Record 6
	Iniziali di identificazione			Numero telefonico					

La tabella consiste di 6 record, (un record è una sequenza di caratteri, o di byte) ognuno dei quali contiene 8 byte. Per ogni record:

I byte 1 - 3 sono un codice di identificazione a 3 caratteri
I byte 4 - 8 rappresentano un numero telefonico a 5 digit.

Ci troviamo quindi ad avere un elenco telefonico per un ufficio di sei persone. Per cercare il numero di telefono di una persona, basta tradurre in rappresentazione ASCII le sue iniziali e confrontare i valori così ottenuti con i primi tre byte dei record della tabella. Quando si trova la corrispondenza, il record è identificato e gli ultimi cinque byte sono il numero telefonico. Eccovi una tabella Esadecimale-ASCII che vi dà i codici ASCII corrispondenti alle lettere dell'alfabeto.

A	41	J	4A	S	53
B	42	K	4B	T	54
C	43	L	4C	U	55
D	44	M	4D	V	56
E	45	N	4E	W	57
F	46	O	4F	X	58
G	47	P	50	Y	59
H	48	Q	51	Z	5A
I	49	R	52		

Ricordate che, come abbiamo visto in un precedente esperimento, la rappresentazione ASCII delle cifre da 0 a 9 va da 30 a 39.
Quindi "AQF" ha come numero telefonico 12306 (è il primo record!).
Il Programma N. 36 legge una tabella di questo tipo e ritorna, come risultato, il puntatore del record i cui primi tre caratteri corrispondono alla stringa di chiave, (confronto), memorizzata nelle locazioni di memoria 0B00, 0B01 e 0B02. Il programma inizia memorizzando nella coppia di registri HL l'indirizzo dell'ultimo record, nella coppia dei registri BC il numero dei record della tabella, e caricando nell'accumulatore il primo byte della stringa di chiave. L'istruzione CPD, e le somme del valore −7 alla coppia di registri HL, confrontano il primo byte di ogni record, dal record N. 6 al record N. 1, per trovare una corrispondenza con il byte 1 della chiave a tre byte. (Notate che ad HL viene sommato −7 al posto di −8, perché l'istruzione CPD decrementa HL). Quando si trova una corrispondenza, bisogna controllare anche i byte due e tre. Questo viene fatto la sequenza di istruzioni che inizia all'istruzione di label CHECK.
Questa sequenza può essere facilmente modificata nel caso si vogliano controllare più byte introducendo un loop; per due soli byte, il loop non è necessario.

APPENDICE I

COME UTILIZZARE LA SGS-ATES Z80-CPU PROGRAMMING REFERENCE CARD

Scopo fondamentale del pieghevole allegato è quello di fornire al programmatore il numero più elevato possibile di informazioni di uso corrente sui codici operativi ed i tempi di esecuzione delle istruzioni, espresse in forma pratica ed immediata.
L'uso della lingua inglese non è casuale: si vuole infatti abituare fin dall'inizio il programmatore ad acquisire dimestichezza con una lingua ormai affermatasi come universale, a cui fa riferimento la terminologia corrente, a partire dagli stessi codici in linguaggio simbolico.

PRIMA FACCIATA

Il pieghevole presenta, nella prima facciata, due elenchi separati di tutte le istruzioni della CPU Z80, con la corrispondenza tra il codice oggetto (OBJ CODE) e la denominazione in linguaggio simbolico (SOURCE STATEMENT), per ciascuna istruzione.
Il primo elenco è ordinato per codici oggetto crescenti, in progressione numerica.
Il secondo è invece disposto in ordine alfabetico, con riferimento alle istruzioni simboliche.

Esempi

Consideriamo le prime tre istruzioni del primo elenco: Z80-CPU INSTRUCTION SORTED BY OP-CODE ("Istruzioni della Z80-CPU ordinate in base al codice operativo") in progressione numerica.

OBJ CODE	SOURCE STATEMENT
00	NOP
018405	LD BC,NN
02	LD (BC),A
....	....

L'elenco inizia con l'istruzione il cui codice oggetto è 00.
Esso prosegue in ordine numerico crescente. Poichè tuttavia al codice operativo 01 corrisponde una istruzione di più byte, questi sono tutti riportati nella colonna OBJ CODE. In questo caso i due byte successivi si riferiscono al numero intero NN che, a titolo esemplificativo, è sempre indicato come 8405 (Ved. tabella EXAMPLE VALUES, in cui sono definiti, con un valore numerico esemplificativo, tutti gli indici e gli interi variabili richiamati dalle varie istruzioni).
Consideriamo ora le prime tre istruzioni del secondo elenco: Z80-CPU INSTRUCTIONS SORTED BY MNEMONIC (istruzioni della Z80-CPU in ordine alfabetico con riferimento al codice simbolico).

OBJ CODE	SOURCE STATEMENT
8E	ADC A,(HL)
DD8E05	ADC A,(IX + d)
FD8E05	ADC A,(IY + d)
....	....

Anche in questo elenco, per le istruzioni che richiamano indici o numeri nel loro codice simbolico (in questo caso: d), questi vengono tradotti in uno o più byte, sempre secondo la già citata tabella EXAMPLE VALUES.

SECONDA FACCIATA

La tabella SUMMARY OF FLAG OPERATION (riassunto delle operazioni sui flag), mostra nella prima pagina le istruzioni che possono modificare i flag ed il modo in cui esse operano. I flag sono definiti nel diagramma FLAGS nell'ultima parte della prima facciata.

Esempio:

INSTRUCTION	C	Z	P/V	S	N	H	COMMENTS
ADD A,s; ADC A,s	↕	↕	V	↕	0	↕	8-bit add or add with carry

Il significato dei simboli è spiegato in calce alla tabella.
In particolare viene usato il simbolo " ↕ " per indicare che il flag in questione viene posizionato in base all'istruzione corrispondente. Nella colonna COMMENTS viene sinteticamente spiegata l'operazione eseguita. Ad esempio, in questo caso, si esegue una somma di 8-bit, rispettivamente senza o con il riporto (carry).

Dopo la leggenda dei simboli che si riferiscono alla prima parte, la tabella prosegue con la descrizione dettagliata di ciascuna istruzione (raggruppata per blocchi di funzioni) per quanto riguarda:

— Codice simbolico (colonna MNEMONIC)
— Operazioni eseguite (colonna SYMBOLIC OPERATION)
— Effetto sui flag (colonna FLAGS)
— Codice operativo (colonna OP CODE 76 543 210)
— Numero di cicli T (colonna NO. OF T CYCLES)
— Eventuali commenti (colonna COMMENTS)

MNEMONIC	SYMBOLIC	FLAGS						OP CODE	NO. OF	COMMENTS	
	OPERATION	C	Z	P/V	S	N	H	76 543 210	T CYCLES		
LD r, r'	r ← r'	•	•	•	•	•	•	01 r r'	4	r, r'	Reg.
										000	B
										001	C
										010	D
										011	E
										100	H
										101	L
										111	A

L'istruzione LD r, r' consiste in un caricamento del registro r col contenuto del registro r'. Essa non dà luogo ad alcuna modifica nei FLAG, ed ha un codice operativo che dipende dai registri r e r', secondo la tabella riportata nella colonna "COMMENTS".
Le cifre da 7 a 0 sotto la dicitura OP CODE indicano l'ordine dei bit all'interno del codice operativo.
Il numero dei cicli di clock, in questo caso, è 4.

APPENDICE II

CALCOLO DEI TEMPI DI ESECUZIONE

In questa appendice viene spiegato come calcola sequenze di istruzioni del microprocessore Z-80. Il tempo di esecuzione è una caratteristica importante di un programma, che deve essere considerata quando si deve scegliere tra metodi alternativi di implementazione.
Si consideri questa sequenza di istruzioni:

```
LD      A,36H
LD      B,49H
OR      B
AND     99H
RL      A
```

Quanto tempo è necessario al Nanocomputer per eseguire queste istruzioni? Per determinare la risposta dovete conoscere la frequenza di clock del Nanocomputer. La frequenza è 2,5 MHz (2,5 Megahertz), cioè, 2.500.000 cicli al secondo. Cioè, ogni ciclo dura

$$\frac{1}{2,5 \times 10^6} \text{ secondi} = 0,0000004 \text{ secondi}$$

Dato che 1 secondo $= 10^3$ millisecondi (ms) $= 10^6$ microsecondi (μs) $= 10^9$ nanosecondi (ns), il ciclo del vostro Nanocomputer è 0,0004 ms, cioé 0,4 μs, cioé 400 ns.
Certi chip Z-80 selezionati possono lavorare a 4 MHz, con un ciclo cioé di 250 ns.
Le tabelle nel pieghevole allegato, descritto in Appendice 1, danno il numero degli stati T, cioé dei cicli di clock, richiesti per eseguire ciascuna istruzione delle Z-80. Così usando queste tabelle, potete eseguire i seguenti calcoli:

Istruzione	Numero degli stati T	Numero di volte che è eseguita	Tempo di esecuzione (μs)
LD A,36H	7	1	7 stati T = 2,8
LD B,49H	7	1	2,8
OR B	4	1	1,6
AND 99H	7	1	2,8
RL A	4	1	1,6

Perciò, il tempo di esecuzione per la nostra sequenza è 11,6 μs.

In questo esempio ogni istruzione viene eseguita una volta sola.

Vediamo ora il caso di un loop di ritardo:

```
        LD A,06H
        LD B,08H
LOOP:   INC A
        DEC B
        JP NZ, LOOP
```

Il tempo di esecuzione di tale sequenza è:

Istruzione	Numero degli stati T	Numero di volte che è eseguita	Tempo di esecuzione (μs)
LD A,06H	7	1	2,8
LD B,08H	7	1	2,8
INC A	4	9	14,4
DEC B	4	9	14,4
JP NZ,LOOP	12 (se è verificata la condizione)	1	4,8
	7 (se la condizione non è verificata)	8	22,4
			Totale: 61,6

Se questo loop di ritardo fosse utilizzato come subroutine, il tempo totale avrebbe dovuto includere anche il tempo richiesto per eseguire sia la CALL iniziale che la RET finale.

CALL	17	1	6,8
RET	10	1	4,0
			Nuovo totale: 72,4

Vediamo ora un ultimo esempio in cui è utilizzata l'istruzione LDIR:

```
LD    HL,0100H
LD    DE,0200H
LD    BC,0010H
LDIR
```

Istruzione	Numero degli stati T	Numero di volte che è eseguita	Tempo di esecuzione (μs)
LD HL,0100H	10	1	4,0
LD DE,0200H	10	1	4,0
LD BC,0010H	10	1	4,0
LDIR	21 (se BC $\neq$ 0)	15	126,0
	16 (se BC = 0)	1	6,4
			Totale 144,4

APPENDICE III

PRECAUZIONI DA ADOTTARE NEL MANEGGIARE DISPOSITIVI MOS

I dispositivi MOS sono estremamente sensibili e possono essere danneggiati da:

Elettricità statica

Inserzione errata negli zoccoli della scheda Nanocomputer.

Nel maneggiare dispositivi MOS, dovrebbero essere osservate le precauzioni seguenti:

1. Assicurarsi di essere elettricamente scarichi prima di toccare il dispositivo. Ciò può essere fatto strofinando le mani su materiale conduttore.
2. Evitare di toccare i piedini del dispositivo.
3. Evitare di mettere a contatto i piedini con qualsiasi materiale capace di caricarsi elettrostaticamente (ad esempio, tappeti di fibra sintetica).
4. Se è necessario trasportare un dispositivo MOS, inserirlo su un supporto conduttore per evitare che tra i diversi piedini si possano avere delle differenze di potenziale.
5. Quando si deve sostituire un dispositivo, assicurarsi di inserire il nuovo dispositivo in modo corretto, in particolare assicurarsi che il piedino 1 si trovi al posto giusto.

APPENDICE IV

ELENCO DEGLI INDIRIZZI ASSOLUTI DELLE LOCAZIONI INDICATE CON NOTAZIONE SIMBOLICA NEL TESTO

Nome simbolico	Indirizzo assoluto
BAUDRT	0F AE
INMODE	0F AB
CONTST	FB 43
MEMTUT	FA DC

APPENDICE V

BIBLIOGRAFIA

1. *The Compact Edition of the Oxford English Dictionary,* Oxford Univ. Press, 1971.
2. Rudolf F. Graf, *Modern Dictionary of Electronics,* Howard W. Sams & Co., Inc., Indianapolis, Indiana, 1977.
3. James Martin, *Telecommunications and the Computer,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1969.
4. Abraham Marcus and John D. Lenk, *Computers for Technicians,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.
5. Microdata Corporation, *Microprogramming Handbook,* Santa Ana, California, 1971.
6. J. Blukis e M. Baker, *Pratical Digital Electronics,* Hewlett-Packard Company, Santa Clara, California, 1974.
7. Donald E. Lancaster, *TTL Cookbook,* Howard W. Sams & Co., Inc., Indianapolis, Indiana, 1974.
8. H. V. Malmstadt, C. G. Enke, e S. R. Crouch, *Instrumentation for Scientists Series, Module 3. Digital and Analog Data Conversions,* W. A. Benjamin, Inc. Menlo Park, California, 1973-4.
9. H. V. Malmstadt e C. G. Enke, *Digital Electronics for Scientists,* W. A. Benjamin, Inc., New York, 1969.
10. J. D. Lenk, *Handbook of Logic Circuits,* Reston Publishing Company, Inc., Reston, Virginia, 1972.
11. A. James Diefenderfer, *Principles of Electronic Instrumentation,* W. B. Saunders Company, Philadelphia, Pennsylvania, 1972.
12. P. R. Rony e D. G. Larsen, *Logic & Memory Experiments Using TTL Integrated Circuits, Book 2,* Howard W. Sams & Co., Inc., Indianapolis, Indiana, 1979.
13. Robert L. Morris e John R. Miller, Editors, *Designing with TTL Integrated Circuits,* McGraw-Hill Book Company, New York, 1971.
14. Charles J. Sippl, *Microcomputer Dictionary and Guide,* Matrix Publishers, Inc., Champaign, Illinois, 1976.
15. Donald Eadie, *Introduction to the Basic Computer,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.
16. Texas Instruments Incorporated, *Microprocessor Handbook,* Dallas, Texas, 1975.

APPENDICE VI

ULTERIORI INFORMAZIONI SUL SISTEMA DIDATTICO NANOCOMPUTER®

Il Sistema Didattico Nanocomputer progettato dalla SGS-ATES è un sistema, uno dei pochi appositamente concepiti a scopo educativo, attraverso cui potete apprendere sia a programmare lo Z-80 che a progettare dei microcalcolatori basati sullo Z-80.
Tale Sistema Didattico è modulare e può essere "fatto crescere" fino a diventare un microcalcolatore completo. E' costituito da tre parti:

Hardware

NBZ80 - Il Nanocomputer. E' un potente microcalcolatore su di una sola scheda, contenente 2K byte di sistema operativo e dotato di una unità di visualizzazione e di ingresso dati. Grazie ad un opportuno Kit di conversione (KNZ80), è possibile trasformarlo in una scheda microcalcolatore SGS-ATES CLZ80, su cui possono essere inseriti un BASIC, un Assembler ecc.

NEZ80 - Scheda per esperimenti. Si interfaccia direttamente con i bus della scheda NBZ80 ed è dotata di una basetta per collegamenti senza saldature, necessaria per effettuare esperimenti di logica digitale e di interfacciamento al microcalcolatore.

NPZ80 - Elegante contenitore metallico per le schede. Contiene un alimentatore completo (± 5 V, ± 12 V).

Kit per esperimenti

Per la realizzazione degli esperimenti descritti nei manuali di istruzione sono anche disponibili un Kit Filatura (K1Z80) e un Kit Componenti (K2Z80).

Nanobook

Nanobook 1 - E' una introduzione ai microprocessori, al linguaggio macchina e al linguaggio Assembler Z-80. Descrive numerosi esperimenti di programmazione che esemplificano i concetti presentati e aiutano il lettore nel corso dell'apprendimento.

Nanobook 2 - E' un'introduzione ai circuiti integrati digitali della famiglia T74LSXX (TTL Low Power Schottky) e ne descrive funzioni e applicazioni tipiche. Numerosi esperimenti esemplificano i concetti presentati e aiutano il lettore nel corso dell'apprendimento.

Nanobook 3 - Tratta in modo esauriente il problema dell'interfacciamento del microprocessore Z-80 con memoria e dispositivi di ingresso/uscita e presenta i dispositivi PIO e CTC della famiglia Z-80. Numerosi e dettagliati esperimenti esemplificano i concetti presentati ed aiutano il lettore nel corso dell'apprendimento.

La relazione esistente tra le varie parti hardware e i diversi manuali di istruzione può essere così riassunta:

	NBZ80	NEZ80	NPZ80	K1Z80	K2Z80
Nanobook 1	X		X		
Nanobook 2		X	X	X	X
Nanobook 3	X	X	X	X	X

Il vostro investimento iniziale non verrà mai sprecato: il sistema è modulare e potete espanderlo via via, passando dal Volume 1 al Volume 3.
Di fatto il Sistema Didattico cresce con voi.
Se desiderate ulteriori informazioni, prezzi, disponibilità, ecc., rivolgetevi al più vicino rappresentante SGS-ATES, citando nella vostra richiesta questo libro.

L. 15.000
(14151)

Z-80 = Trade Mark ...
Nanobook = Marchio ...



La dotto.ssa *Elizabeth A. Nichols* è consulente di sistemi presso la CENT... Reston, Virginia, specializzata nelle applicazioni della microelettronica ai ... energetico ed ambientale. La Nichols ha conseguito il P.h.D. in matematica alla Duke University nel 1974.



Il dott. *Joseph C. Nichols* è consulente di sistemi alla Network Analy... Washington, specializzata nell'analisi e nella progettazione di reti di ... sistemi distribuiti. Ha conseguito il P.h.D. in matematica alla Duke Un...



Il dott. *Peter R. Rony* è professore al Dipartimento di Ingegneria Chimica al Virginia Polytechnic Institute & State University. Rivolge molta attenzione all'elettronica ... ed ai microcomputer da quando essi occupano un ruolo sempre più considere... campo del controllo di processo. E' coautore di molti libri della Blacksburg Co... Education Series TM-e di articoli mensili sull'interfacciamento dei microcompu... appaiono sulle riviste american Laboratory, Computer Design, Ham Radio Mag... rivista tedesca Electroniker, la rivista italiana Elettronica Oggi e altre riviste am... europee.